CWS/CMS

Application Architecture

Version 2.0

June 18, 2004

Table of Contents

Table of Figures

1.0 Summary and Overview

The Child Welfare Services Case Management System (CWS/CMS) architecture is a component-based architecture that combines functionally engineered commercial and custom parts in a modular fashion to create a robust set of functions for the CWS/CMS caseworkers. While externally providing the appearance of a large and complex system, it is actually a set of smaller elementary parts (or components/modules) working together as an integrated system.

This component architecture streamlines the future growth of the CWS/CMS system by both extending the current workstation architecture and adding new technology pillars to the existing system. This allows flexibility by providing an array of technologies and growth in diverse areas, and it even allows the creation of value for sets of stakeholders not originally envisioned.

A review of the CWS/CMS architecture can be summarized as follows:

A. **The architecture is component-based and designed with modular components**, both commercial-off-the-shelf (COTS) and custom (created by IBM specifically for this project). It is engineered in layers for easier maintenance and to allow the components to be isolated for modification or replacement.

B. **The architecture is well suited for continuous and efficient maintenance** because of the modular nature of the program code, which allows the system administrators to isolate and fix individual functions without having to retrace and test every other part of the system.

C. **The architecture is readily extendible** (or extensible) because new applications can be integrated into the existing application services base via documented application programming interfaces (APIs). (An example of this is SOC158, an application added to the CWS/CMS base for non-CWS/CMS placement cases.) New applications can also be added alongside the existing base application, while retaining the single system database and functional view at the same time.

D. **CWS/CMS can benefit from new technologies that the architecture is easily adapted to.** Using Internet technology, new technologies can extend the reach of the system to include new types of user-input terminals and larger populations of stakeholders. These new technologies will be easy to adapt because the current architecture's structure is modular.

A more in-depth description of each of these four points follows.

1.1. The architecture is component-based and designed with modular components

The CWS/CMS application and its underlying technology provide a *component-based* architecture that underpins all functions performed by the system. The following simplified diagrams of this architecture demonstrate this concept.

The diagram below illustrates a high-level depiction of the workstation architecture and shows the supporting components on the workstation for the various CWS/CMS and COTS GUI applications.

High-level View of CWS Workstation Technical Architecture

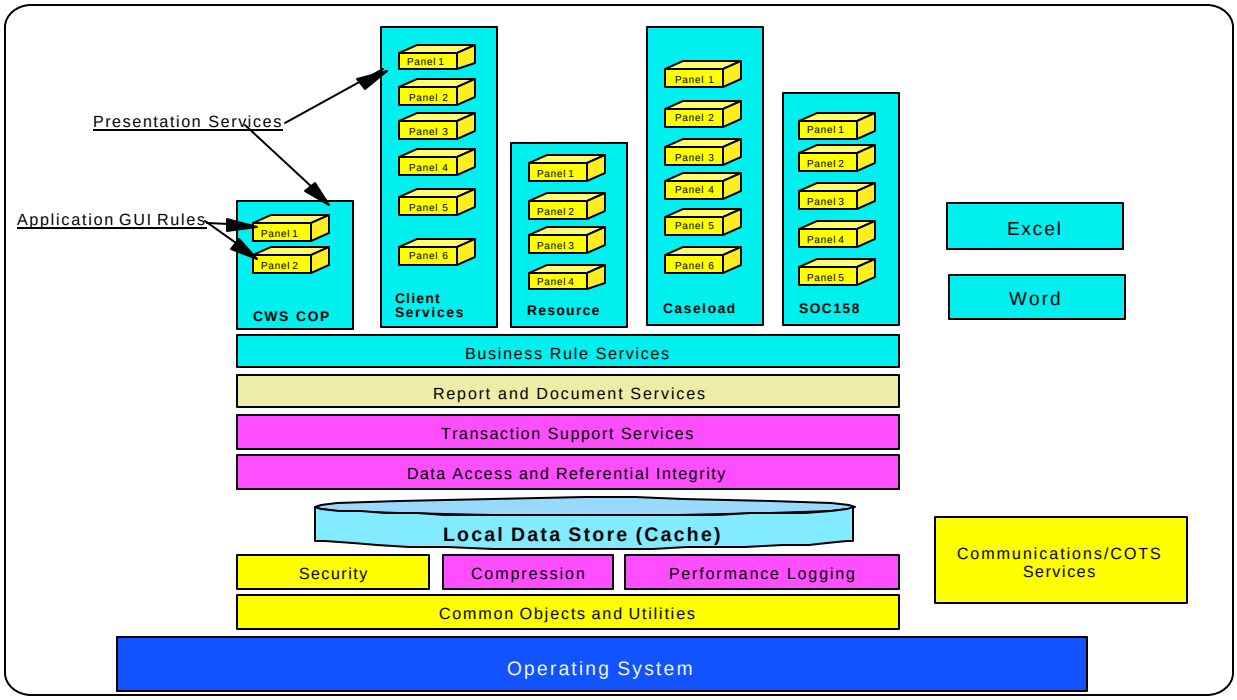


Figure 1: High-level View of Current CWS/CMS Technical Architecture

The diagram below illustrates the various components that inter-operate across the current three tiers — workstation, server, and host. These components provide the foundation and support that is required by an extendible and robust application. Since most of these components are generalized support modules and are managed and updated through the use of code-generation facilities, they make changing the application more efficient.

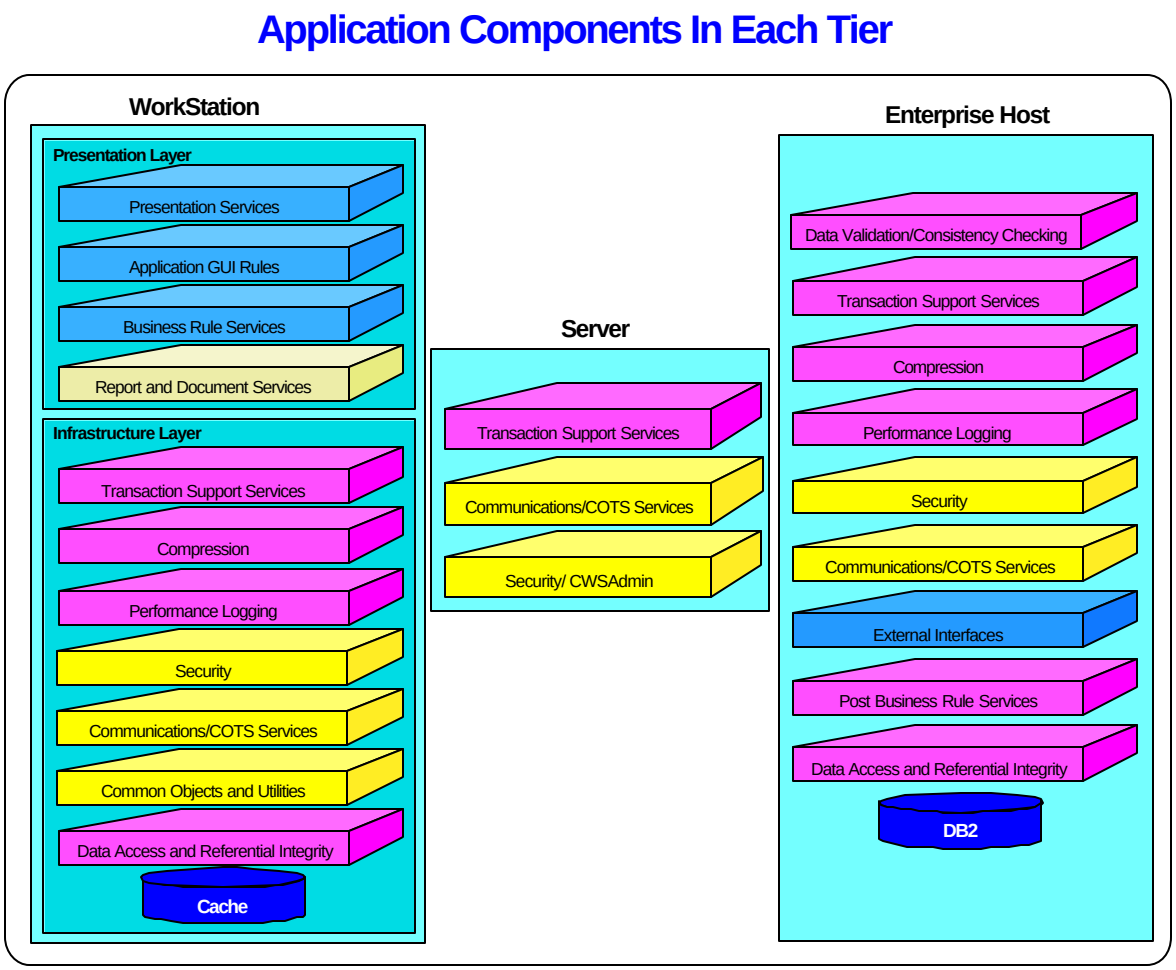


Figure 2: CWS/CMS Tiers and Components

1.2. The architecture is well suited for continuous and efficient maintenance

Because the system is broken down into much smaller modules, maintenance of the system can be performed in small “chunks” or sets of modules. For example, to add a new function to the CWS/CMS application stack, only the application modules that are affected in that stack of programs need be modified. In addition, the “infrastructure” or “system services” modules

normally would not be modified, hence reducing the amount of new programming code required to place the application into production.

In addition, much of the system “backend” is created using generated code and tables that reduce the impact of change. Note that as the system grows, it would be helpful to maintain the system within “towers of function” to reduce the amount of intra-system impact.

Maintenance of existing applications can require both “new functions” and “modifications” to existing functions and can be performed by modifying the existing code and adding new code. The next section discusses the addition of new code as extensions to CWS/CMS.

1.3. The CWS/CMS architecture is readily extendible

Because of the modular structure of the system, it is easily extended in multiple directions. The primary screens are user-directed (technically called “modeless”), and the user determines the navigation from screen to screen, unlike an application-dictated path. This modeless quality allows the system extensions to be built without necessarily impacting system flow or other applications.

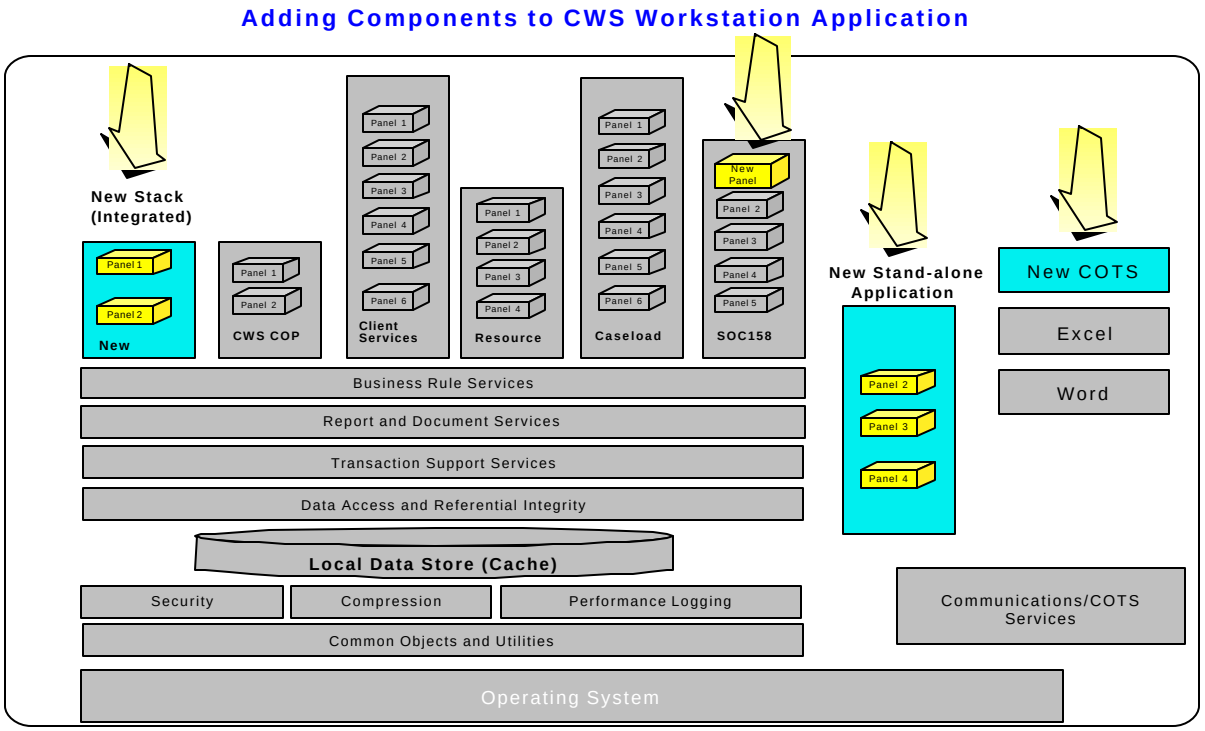


Figure 3: Adding Components to CWS Workstation Application

There are several ways the system can be extended:

- New modules can be added as “parts” to existing stacks of the application. (Refer to the New Stack shown in Figure 3.) This would add a new application to CWS/CMS using the existing infrastructure and new presentation layer code such as Visual Basic 6.0.

- New stand-alone applications can be added to the workstation using only the communications structure of the PC. (Refer to the New Stand-Alone Application shown in Figure 3.)
- New COTS can be added that allow the user to make use of new off-the-shelf personal applications. (Refer to the New COTS shown in Figure 3.)
- New modules can be added to the existing application without impacting the entire existing application infrastructure. (Refer to the New Panel shown in Figure 3.)
- New "subsystems" can be added using only the database as the connection point to the existing system and using none of the existing workstation infrastructure. (Refer to Figure 4.) This provides consistent storage and understanding of the captured data. Because all of the application components and any new application "pillars" use the same basic services, the statewide view of the information is retained.

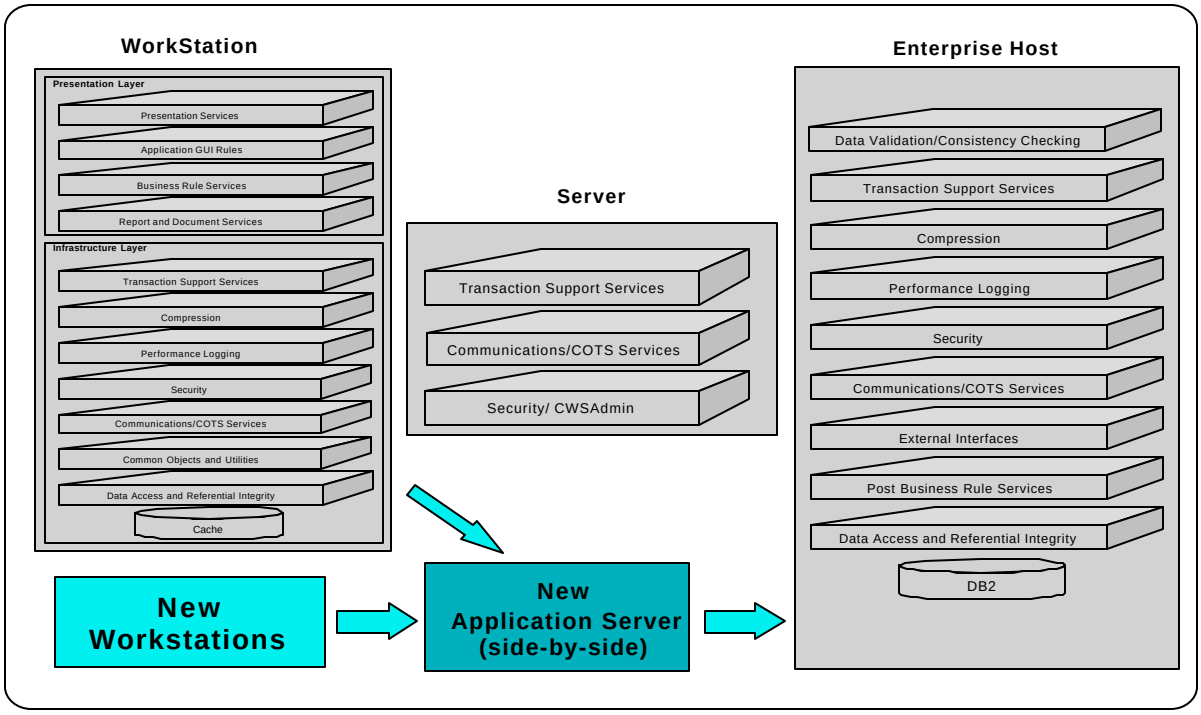


Figure 4: Potential Addition of Subsystems to the CWS/CMS Architecture

1.4. CWS/CMS can benefit from new technologies that the architecture is easily adapted to

Since the CWS/CMS architecture is modular, new technologies can be added to the system in straightforward fashion. In all cases, we recommend adding technologies while preserving the core centralized database of the CWS/CMS system. This means adding the technologies in a way that allows them to continue using the existing services of the central host. These are the benefits gained by employing this method:

- Data is stored once and without duplication, and the rules are uniformly applied in one location, which provides data integrity.
- The state maintains a system that has a single view and single data repository.
- New applications are added faster and without duplication of data because the data access is generated automatically, much of the infrastructure code is reused, and the amount of code to be written and debugged is reduced by pre-existing templates.
- A single location is backed up uniformly with version control and a single point of change management.
- Unsynchronized data with integrity problems is avoided.

The following sub-sections highlight of some alternative technologies that could be used to add new functions to the system and provide value to CWS/CMS.

1.4.1. Web Client Approach

The web client approach for a new function would give authorized individuals an interface accessed with an Internet browser to conduct business within the CWS/CMS system. (Refer to Figure 5). The types of applications best suited to this would be those that do not require the full functionality of the current workstation and would be accessible to those without the standard CWS/CMS terminal. In fact, any state-authorized stakeholder needing to interact with the system would not need special software to use this limited function from anywhere on the Internet or on any terminal.

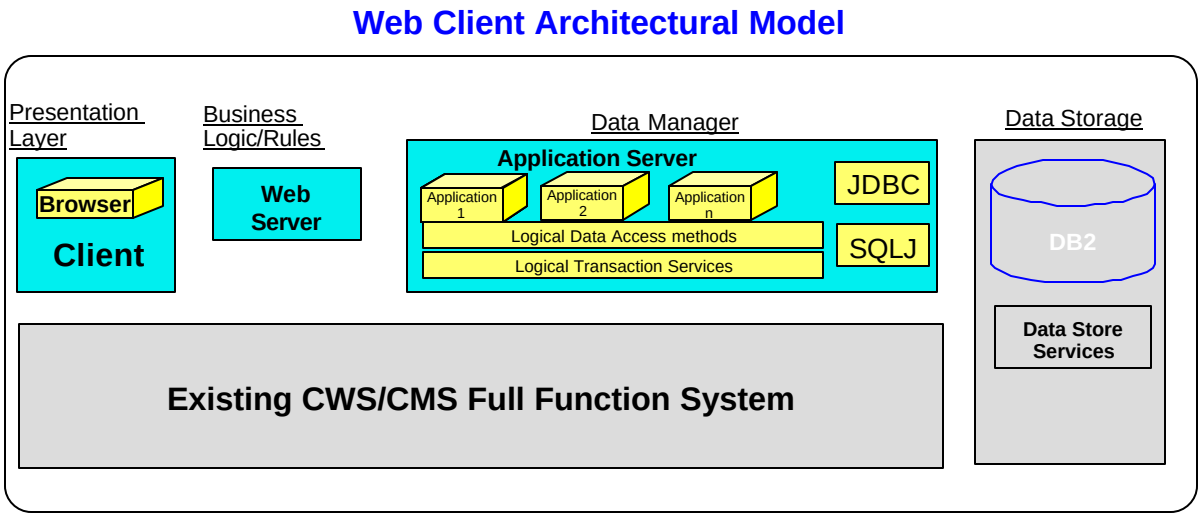


Figure 5: Web Client Architectural Model

1.4.2. Other Recommended Architectures

- **Java client**: This architecture uses Java on the workstation to provide more business rules capability than the thin client; however, it is not as deployable as the browser client is.

- **Distributed SQL**: This architecture uses a fat client and is suited for business information processing, special reporting, and statistical analysis.
- **Non-traditional client**: These architectures include PDA, WAP Phones, and Voice technologies that should be explored to add functionality for individuals who do not have regular or any access to a CWS/CMS workstation.
- **Current and alternative "interfaces"** that can be used to provide updates to or information from the current system.

1.4.3. Architectures Not Recommended

- **Multiple databases**: Due to the replication complexity, architecture that uses additional databases of similar or different technology to build new applications is not recommended.
- **Full CORBA**: Due to application complexity, slower and more costly initial development cycles, and configuration issues, this architecture is not well suited for this application and is not recommended.

1.5. Conclusion

We are confident in the ability of the basic application architecture to support further extensions within the current architecture and to use newer technologies for additional functionality.

2.0 CWS/CMS Architectural Review

In the Summary and Overview section, the review of CWS/CMS architecture produced four main points, the first of which relates to the overall nature of the architecture:

- CWS/CMS architecture is component-based or modular

The other three points are directly related to that inherent architectural structure and include:

- Efficiency of maintenance
- Extendibility
- Adaptability to new technologies

While not addressing each point specifically, the remainder of this document delves into the technical aspects of their impacts.

2.1. CWS/CMS Architectural Components

CWS/CMS is often viewed as a single large application because it has a mission to serve a specific user community. However, CWS/CMS is comprised of many modules, each performing a well-defined set of functions that are combined to form the overall application.

Four major workstation applications are deployed on the CWS/CMS application architecture. These applications are formed from smaller modules and are built on a modular layer of components that provide much of the system function for the applications and form the nucleus of a reusable code set.

Modularity makes the CWS/CMS application architecture a component-based system that is easy to maintain and extend.

It provides reusable service routines, code generation techniques, and a consistent development process to develop functions. The deployment of these components and resulting applications on the architecture's hardware platforms is based on the specific application area as defined by a business need.

Application Components In Each Tier

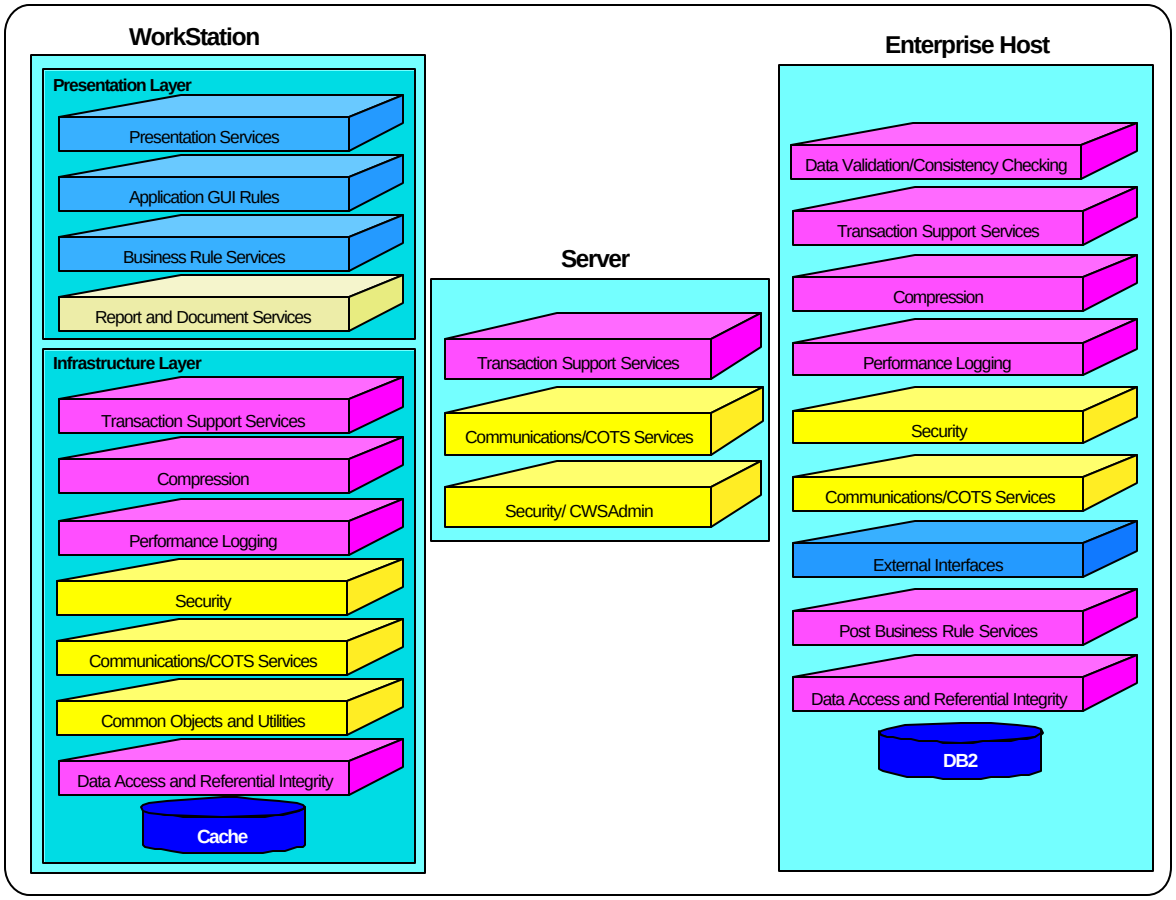


Figure 6: CWS/CMS Architectural Components in Each Tier

New applications are constructed from the architecture's re-usable components or by extending the existing applications. The resulting applications are deployed within the CWS/CMS technical architecture's platforms.

Each platform contains a set of cooperating components that collectively provide the necessary functionality to support the application within an extendible and maintainable environment. The platforms containing the application's functionality are:

- **Workstation**: This platform is where all of the desktop components are located.
- **Server**: This platform is where interim communication components are located.
- **Enterprise host (or mainframe)**: This platform is where the database components are located.

2.1.1. CWS/CMS Workstation Architecture

The CWS/CMS workstation architecture consists of an extendible presentation layer that sits on top of a modular infrastructure layer. The workstation architecture provides for easy extension in the presentation layer with minimal changes required to the underlying infrastructure. The functions within each layer are separated into modular components, each of which contains a set of interfaces that are defined clearly and in detail. This component model provides for a loose functional cohesion that allows functional changes to be made to any single area without a high degree of incidental changes to the other components.

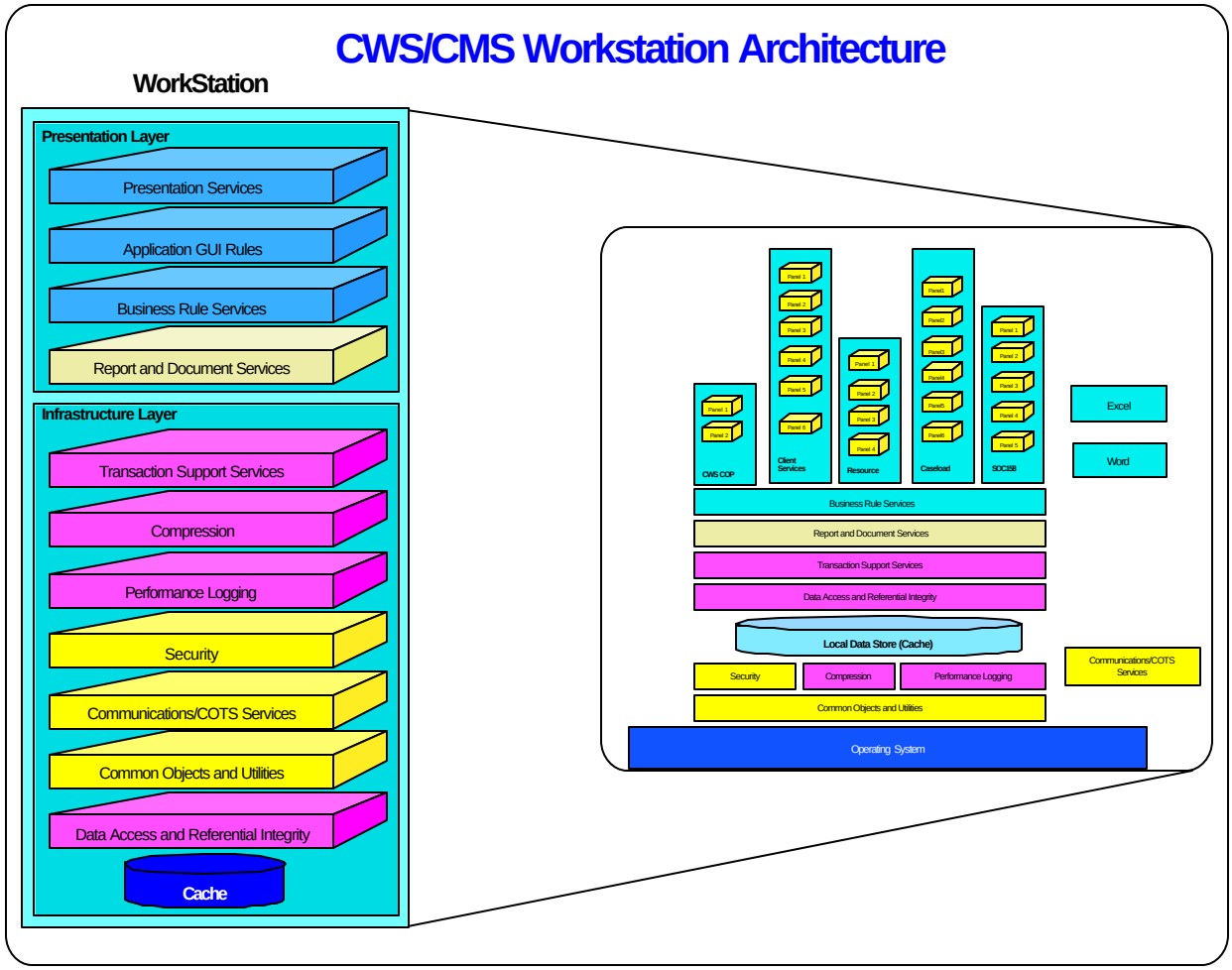


Figure 7: CWS/CMS Workstation Components

The presentation layer consists of both CWS/CMS and non-CWS/CMS applications that share the services provided by the common infrastructure layer. The current applications include:

- CWSCOP
- RESOURCE
- SOC 158
- CLIENT SERVICES
- CASELOAD
- MICROSOFT WORD

The architecture's multi-dimensional extendibility is provided by the ability to add new applications (e.g., SOC158) or the ability to extend an existing application (e.g., the Zippy Referral function that was added to the CWS/CMS application). The architecture also supports non-CWS/CMS applications that need to take advantage of the services provided by the common infrastructure layer. Currently, Microsoft Word is supported, and Microsoft Excel is used for some of the Program Management reports. However, this document only describes the architecture of the CWS/CMS applications.

2.1.1.1. Presentation Layer

The presentation layer of the CWS/CMS applications consists of four major components:

- Presentation Services
- Application GUI Rules
- Business Rule Services
- Report and Document Services

2.1.1.1.1. Presentation Services

The Presentation Services component is the Graphical User Interface (GUI) provided to the user. It is built upon common GUI objects that are mixed and matched to form the notebooks and pages that comprise CWS/CMS applications. The table below details the common GUI components in CWS/CMS.

GUI COMPONENTS COMMON TO CWS/CMS		
Edit Boxes	Picture Boxes	Frames
Option Boxes	Check Boxes	Grids
Tabs	Scroll Bars	Time/Date Controls
Mask Edit	Long Text Boxes	Menu Bars
Status Bars	Labels	Command Buttons

Notebooks and pages are constructed in a Visual Basic 6.0 development environment using common building blocks known as controls. Once a notebook or page is constructed, code is then added to check the business logic, bind the controls to the cache, and present the notebook or page to the user.

The requirements for the CWS/CMS application demand a comprehensive interface to a rich set of functions. However, many of the sophisticated technologies and controls that are now shipped with development products such as Visual Basic were not available at the time the application was in its design and development stages. The standard controls that were available were not adequate to meet the requirements of the system.

The standard controls were either limited in functionality, very slow in performance, or did not offer the level of flexibility the system needed to carry out the requirements. Therefore, many third party and custom controls are used in the presentation services to provide custom functionality and behavior needed by the user. These third party and custom controls provide a way to give the end user the functionality they require -- specialized sorting, customized tabbing, modified colors, etc. -- and are characteristic of the functional richness of a CWS/CMS's "fat client" implementation. The other components of the workstation system (such

as the “Transaction Support Services” that communicate with the back-end database) are heavily dependent on the customized controls and their individual characteristics as a way to provide a communication path with the user interface. The CWS/CMS third party and custom controls include:

Control Name	Vendor/Source	Category	Functionality of the Control in CWS/CMS
COMDLG32	Microsoft	Commercial	Common dialog control
CWSLIST	RDA/In-House	Custom	List and combo box Control
CWSRIBON	RDA/In-House	Custom	The five colored boxes in the application
CWSTAB	RDA/In-House	Custom	Tab Control
DWEASY32, DWSBC32, DWSHK32	Desaware	Commercial	Controls to capture Windows messages for communication with other components in the application
EDITMLV	RDA/In-House	Custom	Virtual Edit Control, which provides a lightweight (low resource) control for data entry.
GTCHKOPT	GreenTree	Modified Commercial	Check box and Option Control
GTDATE32	GreenTree	Modified Commercial	Time/Date Control
GTMASK32	GreenTree	Modified Commercial	Masked Edit Control
GTSCROLL	GreenTree	Modified Commercial	Scroll Control, which allows adding scroll bars to VB windows such as MDIchild forms
MSMASK32	Microsoft	Commercial	Standard masked edit control
PICCLP32	VB	Commercial	Picture Clip Control, which provides an efficient mechanism for storing multiple picture resources. Instead of using multiple bitmaps or icons this control can access images from one large image file.
RICHTX32	Microsoft	Commercial	Microsoft Windows Common Control

Control Name	Vendor/Source	Category	Functionality of the Control in CWS/CMS
SSCALB16, SS32X25	Farpoint Spread	Commercial	Spreadsheet Control, which provides enhanced grid functionality for displaying rows and columns of data.
THREED20	Sheridan	Commercial	Provides several 3D controls. CWS/CMS applications use the 3D panels for status bars.
SPREAD	Spread	Commercial	Spreadsheet Control, which provides enhanced grid functionality for displaying rows and columns of data.
THREED32	Sheridan	Commercial	Provides several 3D controls. CWS/CMS applications use the 3D panels for status bars.

The Presentation Services component also contains common presentation objects that provide the following common services for the applications:

- Notebook navigation and transversal
- Forms loading and unloading
- Error detection and display
- Focus resolution after error detection

These common presentation objects comprise a large portion of each application's code base and do not have to be rewritten when an application is modified or an additional application is added. The Visual Basic language supports common code that is written in an object-like paradigm. Each notebook becomes an object, and the methods or functions associated with each object are basically the same and can be re-used as a model for new function.

2.1.1.1.2. Application GUI Rules

Customer business logic is unique to each application and can be verified at the client, the host, or an intermediate level. In the CWS/CMS application, business logic is verified in three places:

- At the Workstation, early verification using the Presentation GUI Rules
- At the Workstation, late verification using Business Rules Services
- At the Host, verification using Data Validation for Applications

The idea behind the GUI rules (early validation) is to prevent input conflicts prior to delivering data to the host. This speeds up the input process for the end user and frequently eliminates the delayed response times that are common in large systems. An example of this type of rule is a grid control that is read-only until a user enters data in a textbox control. Once a field is entered in the textbox, the grid becomes writeable. In the GUI application layer, only early verification is performed.

Late verification is performed on the data using Business Rule Services in the presentation layer before it is saved to the database. At the host, verification is performed as the data is stored into the host DB2 database using the Post Business Rule Services.

2.1.1.1.3. Business Rule Services

When a user has completed work and starts to save it to the host database, the workstation presentation layer performs late business logic processing before it sends the data to the host. The code loops over each open notebook object and checks for business rule violations such as non-alphabetic characters in a name field. Calls are then made to the cache database component to verify mandatory data fields and inter-notebook referential integrity. If an error occurs during the late business rule verification, the notebook causing the error is re-opened and focus is placed on the field in question.

2.1.1.1.4. Report and Document Services

The Report and Document Services component enables the users to generate local reports and documents by interfacing the presentation layer with the COTS presentation components. Currently, Microsoft Word is used for the presentation of the local reports and documents. Documents are saved and stored to the database, but reports are not saved. The following diagram illustrates the generation of local reports and documents.

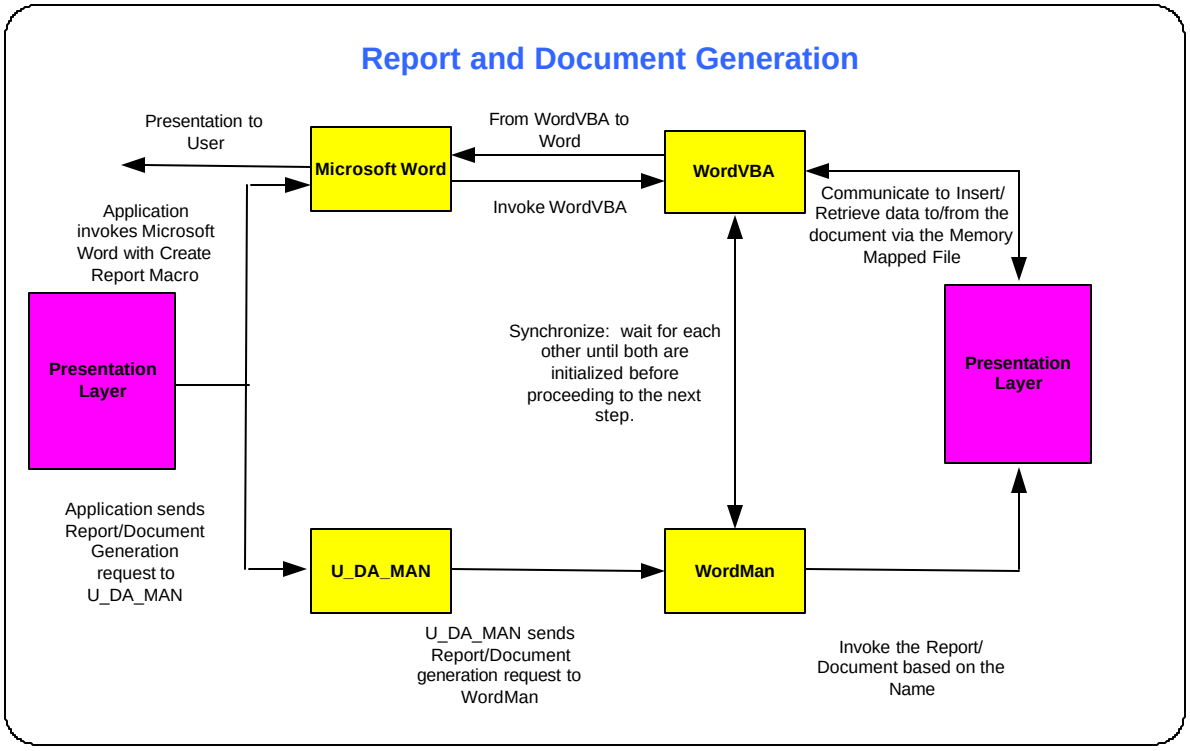


Figure 8: Overview of Report and Document Generation

2.1.1.2. Workstation Infrastructure Layer

The workstation infrastructure layer consists of the following components:

- Transaction Support Services
- Compression
- Performance Logging
- Security
- Communications/COTS Services
- Cache
- Data Access and Referential Integrity
- Common Objects and Utilities

2.1.1.2.1. Transaction Support Services

CWS/CMS transactions originate at the workstation presentation layer and represent the user's need to review and modify child welfare services information. A transaction is a combination of a *request* issued from the workstation and a *response* returned by the enterprise host and involves transfer of information through all the three architectural tiers. The transaction data traveling between the workstation and the host is organized into data blocks called transaction packets as defined by the Transaction Packet Descriptors (XPDs).

The Transaction Support Services components on the workstation, server, and enterprise host are responsible for receiving transaction requests, performing necessary processing, and delivering the responses. They utilize the services of other components such as Compression and Communications/COTS Services for the successful completion of the transactions. The following are the main steps in transaction processing:

- Client Submission and Packaging
- Submission Transmission Processing
- Data Processing (Transaction Execution)
- Response Transmission Processing
- Unpackaging and Data Delivery

The Transaction Support Services component on the workstation is responsible for the *Client Submission and Packaging* and the *Unpackaging and Data Delivery* steps. It receives the transaction requests from the presentation layer, examines each transaction request, and determines the information it needs to construct the XPD for transmission to the enterprise host. It then reads the appropriate data from the workstation cache, packetizes the data into the correct XPD format, compresses the data using Compression component, and submits the transaction to the host using Communications/COTS services components. When the host executes the transaction and returns data to the workstation, the Transaction Support Services component decompresses the data using Compression component, examines the returned XPD format, de-packetizes the data and stores the data objects in the workstation cache.

The diagram below illustrates the transaction processing flow in CWS/CMS.

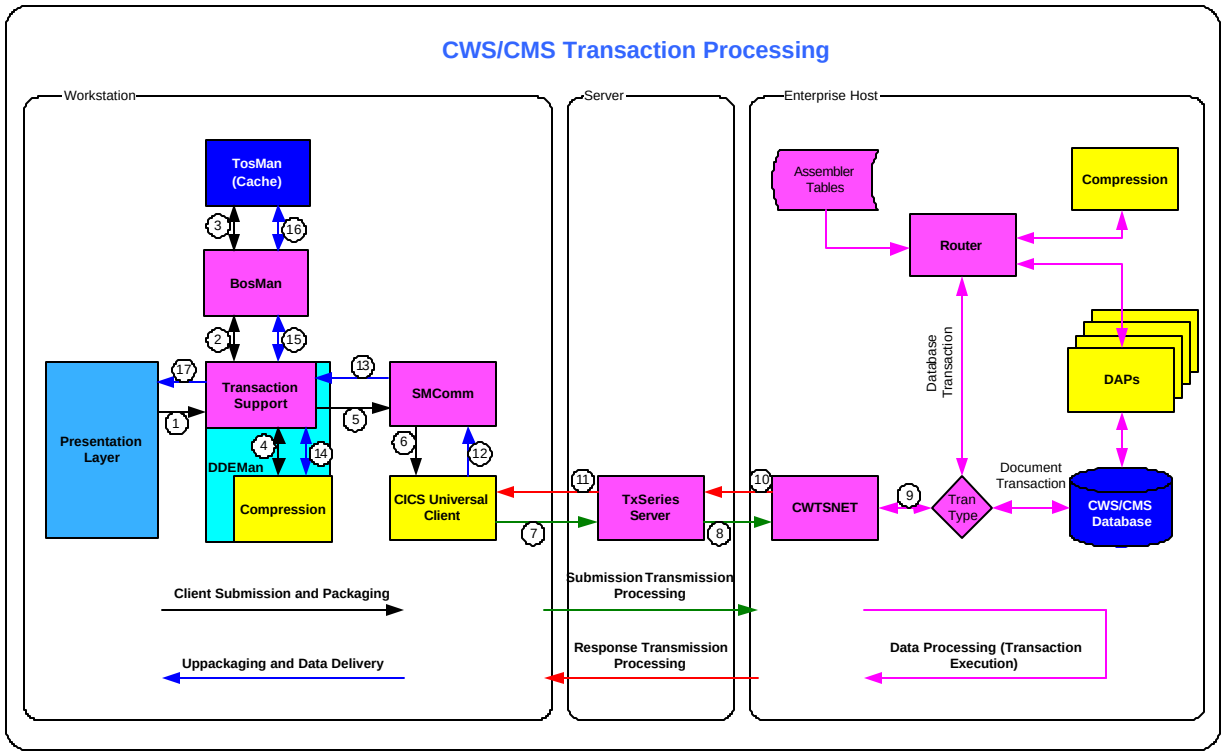


Figure 9: CWS/CMS Transaction Processing Flow

A description of the Transaction processing steps is provided below:

Client Submission and Packaging

1. The presentation layer calls the Dynamic Data Exchange Manager (DDEMan), a Dynamic Link Library (DLL) of the Transaction Support Services component on the workstation, to execute a transaction in response to a user action that requires data access from or data modification to the enterprise host database.

2. DDEMan uses the input key provided by the presentation layer and calls the Batch Output System Manager (BOSMan), a DLL of the Transaction Support Services component on the workstation that provides packetization/de-packetization services, to construct the XPD.

3. BOSMan calls the Transaction Optimization System Manager (TOSMan), the workstation Cache component, to construct the data packets.

4. DDEMan utilizes the Compression component on the workstation to compress the XPD. It is to be noted that the Compression component is physically contained in the DDEMan DLL.

5. DDEMan calls the Socket Manager Communications (SMComm), a DLL in the Transaction Support Services component on the workstation that provides interface to

the workstation's Customer Information Control System Universal Client (CICS Client), to initiate the transaction.

6. SMComm reads the XPD, formulates the transaction, and calls CICS Client to execute the transaction.

7. The CICS Client submits the Transaction Request to the TxSeries Server, the Transaction Support Services component on the Server.

Submission Transmission Processing

8. The TxSeries Server receives the notification to execute the transaction. Since the transaction is defined to exist remotely on the enterprise host, the TxSeries transmits the transaction to the host.

Data Processing (Transaction Execution)

9. The Transaction Server Network Interface program (CWTSNET) receives the transaction request and collects the input transaction data. It examines the input transaction data and identifies whether it is a document transaction or a database transaction. The CWTSNET directly processes the document transactions. It calls the Transaction Router Program to process the database transactions. The Transaction Router de-compresses the input transaction data using the Compression component on the enterprise host. It dynamically builds the execution sequence of the Data Access Programs (DAPs) based on the transaction definitions stored in the Assembler Tables. It calls the DAPs in sequence corresponding to the transaction and builds the transaction response. The transaction response data is compressed using the Compression component prior to the delivery.

Reponse Transmission Processing

10. CWTSNET delivers the transaction response to the TxSeries on the Server.

11. TxSeries Server returns the transaction response to the CICS Client on workstation.

12. The CICS Client on the workstation returns the transaction response to SMComm.

Unpackaging and Data Delivery

13. SMComm collects the transaction response, constructs, and returns the XPD to DDEMan.

14. DDEMan utilizes the Compression component on the workstation to de-compress the XPD.

15. DDEMan calls BOSMan to de-packetize the XPD.

16. BOSMan de-packetizes the XPD and call TOSMan to store the data objects in the Cache.

17. DDEMan notifies the presentation layer that the transaction is complete.

2.1.1.2.2. Compression

The Compression component on the workstation provides data encryption, data compression, data decryption, and data decompression services to the Transaction Support Services

component. The Transaction Support Services component calls the Compression component after constructing a transaction packet. The Compression component encrypts and compresses the data, then writes the data to a temporary file. The data in the file is transferred to the host using the CICS client COTS product. When transaction response data is returned to the workstation from the host it is in an encrypted and compressed form. The Transaction Support Services component stores the transaction response in a temporary file and calls the Compression component, which then decrypts and decompresses the transaction response. The decrypted and decompressed data is then passed to the Transaction Support Services component to be de-packetized and placed in the workstation cache.

Since the Compression component is an integral part of the transaction processing in CWS/CMS, the workstation Compression component is physically placed in the DDEMan DLL.

2.1.1.2.3. Performance Logging

The Performance Logging component provides a means of capturing statistics pertaining to the performance of the CWS/CMS transactions. Every time workstation executes a transaction to access data from the enterprise host database, the Performance Logging component collects the performance information regarding the transaction including the transaction type, elapsed time and the compressed and uncompressed size of the transaction response received from the host. This performance information regarding each individual transaction is recorded in the host database, in the PERFLOGT table. The workstation creates a separate thread and executes an asynchronous transaction to store the performance data on the host, which ensures that the users are not kept waiting while the performance data is being logged.

The performance logging transaction is similar to the other database transactions except that it is initiated by the Transaction Support Services component on the workstation and is processed by a separate Performance Logging component on the host. The performance data collection services are part of the DdeMan DLL.

DDEMan collects the performance data during the execution of a CWS/CMS transaction. At the completion of the transaction, DDEMan then makes a call to the PipeMan to record the performance data. This call to PipeMan is made to determine the server operating environment. In the current Windows 2000 server environment, PipeMan simply returns the call indicating that SMComm should be used for logging performance data. DDEMan will then record the performance data using SMComm, which is responsible for running transactions through Transaction Server and on up to the host. The result of this is that the performance data is sent to the host and recorded in a DB2 table, PERFLOGT.

The following diagram illustrates the performance logging.

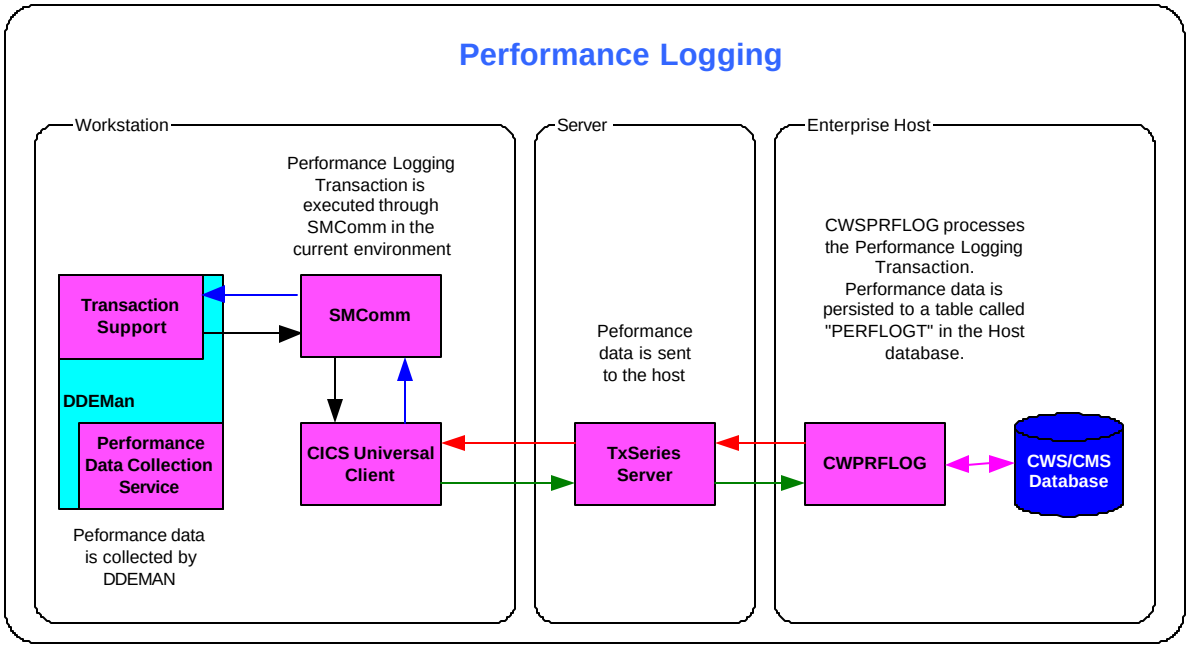


Figure 10: An overview of performance logging

A Distributed Program Link (DPL) is the mechanism used for sending the data from the Workstation to Host. The Workstation issues a CICS ECI call to execute a remote program in CICS on the Server. In essence, the ECI call will link to a program on the Host. The performance data is sent in the COMMAREA, and finally, the CWPRFLOG program on the host formats the data and stores it in a DB2 table. (Refer to *Appendix C* for the format of the COMMAREA and the structure of the DB2 table.)

If there is an error while storing the data in the DB2 table, the data is stored in a VSAM data set. Every night, a program is run to read the data from the VSAM data set and store the data in the DB2 table. After this batch job, another program unloads the previous day's data from the DB2 table into a data set kept on a DASD that is shareable between State of California and EPDM systems. A scheduled program on the EPDM system reads this data and uploads the data into the EPDM database.

Since the CWS/CMS system is 24 x 7 system, it was necessary to design the DB2 tables so that maintenance done on the table (e.g., deleting old rows) does not make the table unavailable. To meet this requirement and make queries on the table run efficiently, the DB2 table is stored in a table space with 31 partitions. Each day's data is stored in a separate partition, which makes it possible to reorganize the table without taking it offline.

The nightly based batch jobs on the host unload the performance data from the DB2 table into a data set used for reporting. Logging the data centrally has several advantages. It allows the use of Data Warehousing tools to analyze the trends and eliminates the need to collect the data from the multiple county servers.

2.1.1.2.4. Security

The Resource Manager workstation application provides the user interface for the county office administrators to locally administer staff registration tasks, and when needed, add and remove

user authorities and security privileges. This improves the application's availability and removes the constraints involved in the central administration of a large, distributed user community. These components also support CWS/CMS password management services. Refer to the *Security/CWSADmin* section for additional information regarding the user administration.

2.1.1.2.5. Communication/COTS Services

The communications in the workstation tier fall into two categories: the inter-process communication (or inter-layer communication) and the inter-tier communication, (i.e., the communication between the workstation and server). The workstation infrastructure components U_DA_MAN and APIMAN provide the mechanism to transport the application programming Interface (API) calls between the presentation layer and the infrastructure layer. The communication between the workstation and the server for the application transaction execution utilizes the CICS Client COTS component.

2.1.1.2.6. Cache

The workstation cache component can be thought of as a miniature in-memory database that houses data retrieved from the enterprise host database. The cache or the workstation database contains all of the information relevant to a user's current task, and the data stored in it is available for use by the CWS/CMS applications. All access to the database data by the CWS/CMS applications is through the cache, which acts as a proxy between the client and the host database.

The cache is essentially a piece of software within the CWS/CMS client application known as Transaction Optimization System Manager (TOSMan). The cache contains structures for all of the tables and views in the host database; when the host makes a change to a table or view, the cache requires the same update. A set of automated tools is used to keep the cache and the host database in synch. The tools automatically generate code for the cache changes. No hand coding is needed to add or change a table or view in the cache database.

Cache allows the CWS/CMS applications to perform host database transactions rather than interactive data retrieval and thus facilitates transaction optimization by reducing the number of required database transactions. After a user logs on, rows of data -- based on a predetermined set of rules -- are imported from the database into the client workstation and are stored in the cache. From that point onwards, database transactions are executed as needed in response to user's request and the data returned by the transaction is stored in the cache.

2.1.1.2.7. Data Access and Referential Integrity

As mentioned before, all access to the database data by the CWS/CMS applications is through the cache, which acts as a proxy between the client and the host database. The Data Access component provides macro-type interfaces into the cache database for common operations such as joins and selects. In addition, the data access layer provides a method to register data displayed in controls to the cache database. Once registered, the cache will notify the control when the data changed and the data will be updated on the user's screen. This function is especially useful when the same data is displayed on more than one notebook, and it allows the user to change data on one notebook and have it reflected on another notebook.

The workstation cache also provides for inter-object data integrity checking. It keeps a list of which objects have changed and determines if a change to one object impacts another one. If so, it checks both objects before sending the data to the host. This helps avoid sending data conflicts to the host and having the transaction returned to the workstation because of an error.

By saving the data in the cache database, the system provides many functions automatically. It maintains a shared area for data used across the multiple presentation applications. It maintains a list of data objects that have changed. It controls references to database objects that need to be checked for referential integrity. These functions allow the user to save the cache data to a local file (in the CWS/CMS application, the **Save Locally** command) and then pull the data up from the local file at a later time. Once the data is restored, the user can continue working or save it to the host database.

2.1.1.2.8. Common Objects and Utilities

The workstation infrastructure layer contains several common objects and utilities to provide services such as spell check, collect application trace data and debug information. The error processing and error message handling is performed through common objects. In addition, the code for several common business rules is contained in common objects.

2.1.2. CWS/CMS Server Architecture

The local server platform of the CWS/CMS application architecture contains the following application architecture components:

- Transaction Support
- Communications/COTS Services
- Security/CWSAdmin
- CWSAdmin

The local server components are deployed to assist in supporting the end-to-end application. In addition to providing application functionality, the local server platform provides system management functionality for network management and software distribution.

2.1.2.1. Transaction Support

Transaction support on the server platform utilizes the following COTS products:

- CICS Transaction Gateway
- TxSeries Server

The server platform provides transaction gateway services to the CICS universal client software on the desktop. TxSeries server provides the transaction support services for the regular database and document transactions initiated by the user. The user administration transactions and the initial user authentication transactions are executed through the CICS Transaction gateway. These products are extendible and support a variety of desktop and server platform operating system choices, and they can also be configured to support different LAN and WAN network communication protocol choices such as NetBIOS, TCP/IP, and SNA.

2.1.2.2. Communications and COTS

The local server platform's Communication components consist of primarily commercial off the shelf (COTS) software. This COTS software (CICS Transaction Gateway, TX Series, and Communication Server) provides the connectivity between the desktop presentation and the Enterprise Host central database persistence.

2.1.2.3. Security/CWSAdmin

The local server platform supports the application security requirements by providing service components for RACF user administration to the application. CWSAdmin component is responsible for user administration and user authentication (log on) services of the CWS/CMS application. The user administration and logon components of CWSAdmin can be viewed as the medium through which users get signed on to use the CWS/CMS application.

2.1.2.3.1. User Administration

CWSAdmin handles user log on to the LAN and to the CWS/CMS application. It also enables the administrator to add, delete, and reset the user passwords. The user IDs are created and maintained for Windows 2000 and RACF. These user administration features are performed from the CWS/CMS Resource Management program.

There are the ten main functions of CWSAdmin pertaining to user administration:

- **RACF Signon**: CWSAdmin uses RACF Signon to check the validity of a user.
- **RACF Password Change**: Password change at the RACF level.
  NOTE: Unlike Password Reset, a password change does not require administrative privileges since the users themselves are able to change their password as they wish.
- **RACF ID Creation**: Creating a new user ID at the RACF level
- **RACF Password Reset**: Password reset at the RACF level
- **RACF ID Deletion**: Removing a RACF ID from the host
- **RACF ID Modification**: Relocating a RACF ID between groups
- **LAN ID Creation**: Creating of a new user ID in Windows 2000 domain/Active Directory
- **LAN ID Password Change**: Changing a LAN password in Windows 2000 domain
  NOTE: Unlike Password Reset, a password change does not require Administrative privileges since the users themselves are able to change their password as they wish.
- **LAN ID Deletion**: Deleting a LAN user ID from the Windows 2000 domain

2.1.2.3.2. User ID Settings

A single user ID can be attached to a CWS staff person. A user ID can be up to seven characters long, and to make it unique, it is defaulted to the first five characters of the staff person's last name, their first initial, and another letter. User IDs are stored in the USER_ID entity. If a unique user ID can not be found, an error is generated. There are five different levels of authority a staff person may have: *User, Office Administration, County Administration, State Administration, and Global Administration*. This information is maintained for the user in the STAFF_PERSON_PRIVILEGE entity. The level of authority determines whether a staff person can administrate users in an office.

CWS/CMS users fit into five categories:

- **User.** This is the normal mode for a caseworker. They have access to all user-level applications and the appropriate case data.

- **Office Administrator.** This authority is one level above User, and it is intended for an individual at a County office. Office administrators can add, modify, and delete any user in their office, but they cannot make changes to users in other offices or counties.
- **County Administrator.** This authority is one level above Office Administrator. They have the same authority granted to Office Administrators, and they can add, modify, and delete any user in their county. They cannot make changes to users in other counties.
- **State Administrator.** This authority is one level above County Administrator. They have the same authority granted to County Administrators, and they can add, modify, and delete any user in all counties.
- **Global Administrator.** The Global Administrator is the highest level of authority for CWS/CMS, and its authorization spans all counties. Global Administrators have the ability to create and maintain users, Office, County, and State Administrators.

2.1.2.3.3. Technical Overview of User Administration

User Administration can take place from any workstation on the network. This allows a super user ID with administrative authority to be configured in RACF and on every LAN Domain Controller. User administration is part of the CWS/CMS Resource application. User creation and maintenance for LAN accounts is performed by a Java application that interacts with the workstation using TCP/IP sockets. RACF user administration is done from CWS/CMS Resource by making a call to the host. Changes to the Windows 2000 domain controllers are made through a set of scripts called through the Java application.

2.1.2.3.4. User Administration Tasks

- **CWS/CMS Logon.** When the CWS Control Panel (CWSCOP) is started, the user is prompted for their user password, and CWSCOP validates the user ID and password with the host (RACF). CWSCOP opens a named pipe or socket to the application server running the CWSAdmin Program. CWSAdmin connects to RACF using the APPCPEM API and a Communication Server LU6.2 connection to verify user ID/password. If this fails, access to CWS/CMS applications will not be allowed.
- **Password Change.** Password changes are made using CWSCOP. CWSCOP opens a named pipe or a socket to the application server running the CWSAdmin program. To change a user password, CWSAdmin connects to RACF using the APPCPEM API and a Communication Server LU6.2 connection. A password change only works in the current domain; if the user is using a resource in a different domain, they will get an access denied error the next time they attempt to log on. However, passwords can be reset for users in another site within the same county (i.e., a different Organization Unit within the same domain). It is important to note that synchronization between the RACF and the LAN password is not always guaranteed when an error occurs during the password change process. Resolving this issue requires intervention by the system administrator, who should reset the password again or manually change the password on the LAN so that the two passwords will match.
- **User ID Creation.** Creating new users will be done in the following order: CWS, RACF, and LAN. If a user creation failed on a system, the user will have to be manually added to the system.

- **User Deletion.** The RACF user ID will be deleted and the staff person information will be end dated. The user ID will be removed from RACF, followed by the removal of the user from Windows 2000 Active Directory.

2.1.2.3.5. RACF Validation

RACF is the primary application security system. It is in the RACF that all applications and transactions are verified against various user lists. Each transaction request is checked against a table of authorized users to ensure the security of the system.

In addition, the custom security system described herein checks with RACF when a user logs on to CWS/CMS application to validate their user ID and password. RACF also insures that the passwords are of the correct type and fit other system-defined parameters such as age, length, etc. No transaction or application program can be executed nor can data be retrieved from the CWS/CMS database unless it is authorized by RACF. DB2 also uses RACF to prevent unauthorized access to the database system. This further ensures that the data being accessed or updated is consistent with the users' privileges.

When the user starts the CWS/CMS Control Panel, they enter their user ID and password. Once entered, a RACF sign on check is immediately made. If successful, the system returns information regarding the last time the user logged on and when their password expires. This information is displayed in the CWS/CMS Control Panel information page. If the RACF sign on check fails, it can return a variety of errors. In most cases, the displayed error is based on the return code. If the RACF sign on indicates the user's password has expired, a change password dialog is displayed. The user is then required to enter their current password and a new password which issues a RACF change password call. If successful, the user's password is changed at the host as well as on the network. If it fails, an appropriate error message is displayed to the user.

Figure 11 provides an overview of the workflow for the RACF workings in the CWSAdmin architecture.

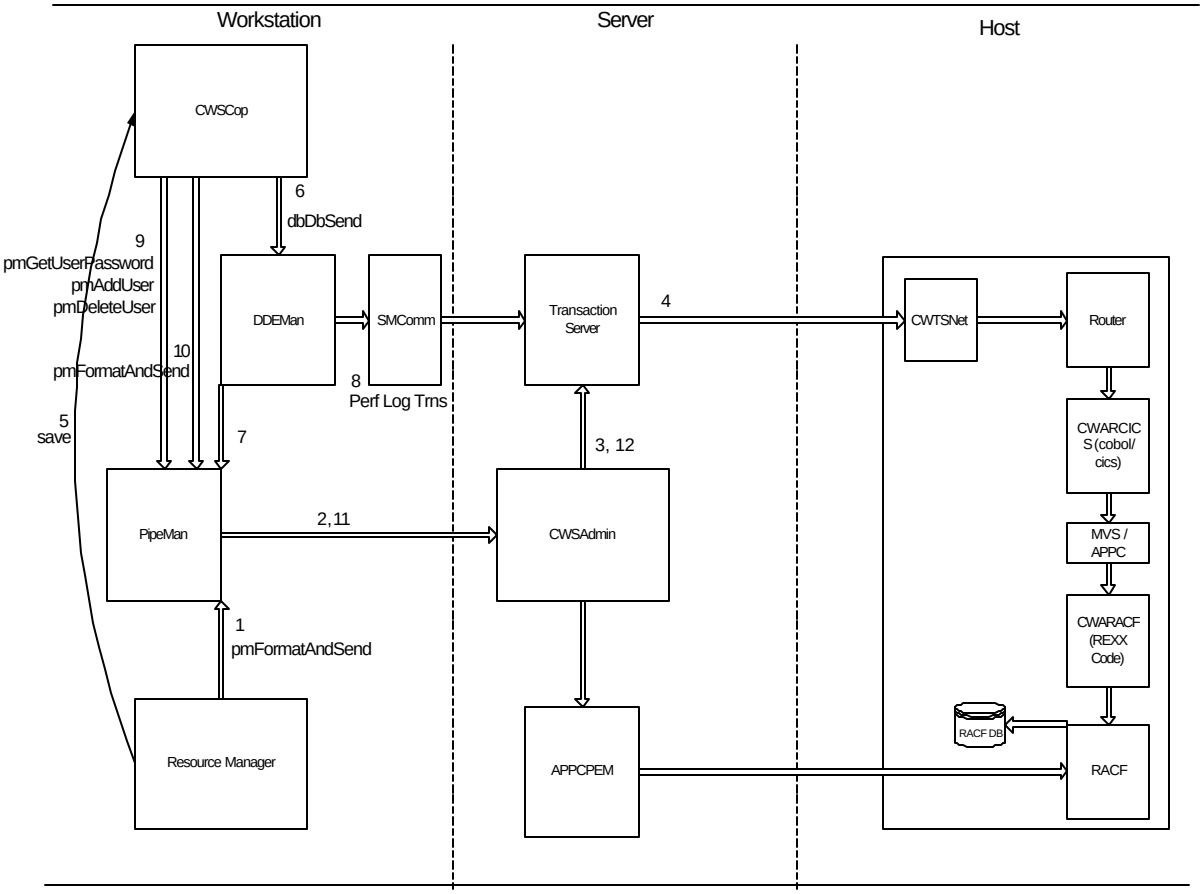


Figure 11: Overall CWSAdmin Architecture

- Using Resource Manager, an administrator creates a staff person and requests that a RACF ID be created.
- PipeMan calls BOSMan to construct an XPD (RACF@AddUser – Y092) and then sends the resultant XPD to CWSAdmin.
- CWSAdmin runs the Y092 transaction, sending the XPD generated in step 2 to the host. The transaction is started using the CWSADM ID, the host RACF administrator.
- Transaction Server function ships the Y092 transaction to the host and executes it there. The host does its RACF processing.
- A save to database is started once the administrator completes the work in Resource Manager.
- CWSCOP calls DDEMan to perform the normal save to database for the Resource Manager application (Y088).
- Once the transaction completes, DDEMan calls pmPerformanceLog to log the transaction performance statistics.

- If the environment is determined to be Windows 2000, DDEMan sends the performance statistics to the host via SMComm.
- CWSCop calls pmGetUserPassword, pmAddUser, and/or pmDeleteUser (as appropriate) depending upon the actions taken by the administrator in Resource Manager.
- As the save to database processing continues, CWSCop calls pmFormatAndSend.
- PipeMan calls BOSMan to construct an XPD (RACF@ModifyUser – Y093) and then sends the resultant XPD to CWSAdmin.
- CWSAdmin runs the Y093 transaction, sending the XPD generated in step 11 to the host. The transaction is started using the CWSADM ID, the host RACF administrator. Transaction Server function ships the Y093 transaction to the host and executes it there. The host does its RACF processing.

Lastly, the table below demonstrates the formatting of data oriented in the CWSADMIN Transaction Format for RACF communication:

	Action	Pipe Format
A	ADD USER	A|DOMAIN|USERID|PASSWORDE|USERTYPE|USERNAME|LOGON|HOMESERVER|HOMEDIR|DIRSIZE|MAILSERVER
C	CHANGE PASSWORD	C|DOMAIN|USERID|PASSWORD|NEW-PASSWORD
D	DELETE USER	D|DOMAIN|USERID
P	PASSWORD RESET	P|DOMAIN|USERID|PASSWORD
R	RACF SIGNON	R|DOMAIN|USERID|PASSWORD

2.1.2.3.6. User Administration and Windows 2000

As mentioned earlier, all modifications to user ID settings are persisted at the RACF and LAN level. The LAN is a Windows 2000 network with an Active Directory installed.

Active Directory is the directory service used in Windows 2000 and is the foundation of Windows 2000 distributed networks. Active Directory provides secure, structured, and hierarchical storage of information about the interested objects in an enterprise network, such as users, computers, services, and so on. The directory provides rich support for locating and working with these objects. Therefore, it is best to accomplish various user administration tasks by using the Active Directory and a set of abstraction objects interfacing with Active Directory called the Active Directory Services Interface (ADSI). ADSI makes it easier to perform common administrative tasks such as adding new users, managing printers, and locating resources throughout the distributed computing environment.

ADSI is a set of COM interfaces used to access the capabilities of directory services from different network providers in a distributed computing environment and to present a single set of directory service interfaces for managing network resources. Administrators and developers can use ADSI services to enumerate and manage the resources in a directory service, no matter which network environment contains the resource.

There are three script files that provide the core user administration functions that are required by the CWSAdmin, such as the addition and deletion of the users, changes in password, and other changes to user attributes. These script files use ADSI to modify the user attributes directly in the Windows 2000 Active Directory. The Java program client CWSAdmin calls the script files and passes the appropriate parameters and the COM object. The CWSAdmin program is a client-server application that resides on the application servers, and it allows users with administrative privileges to change user settings from virtually any workstation within the network. Figure 12 illustrates the overall architecture of user administration on Windows 2000 domain structure and Active Directory.

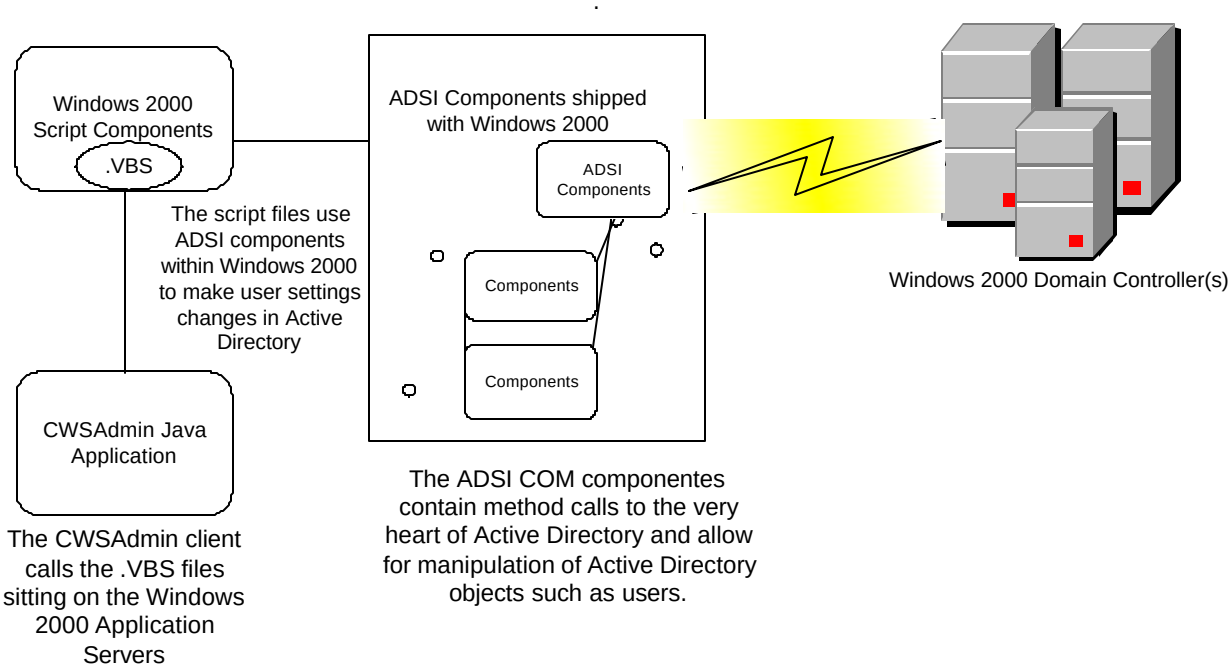


Figure 12: User administration on Windows 2000 servers

2.1.2.3.7. Physical topology of CWSAdmin

CWSAdmin is accessed through the Resource Manager component in the CWS/CMS application, which sits on the client workstation. CWSAdmin resides on the individual application servers in each county. Upon user validation by Resource Manager, CWSAdmin processes LAN user administration requests by using the ADSI COM object. The ADSI classes complete the needed user administration tasks, after which the appropriate changes are made to Active Directory.

2.1.3. CWS/CMS Enterprise Host Architecture

The Enterprise host platform provides the central operating system services that support the application architecture's database. This host manages the transaction load provided by the distributed user community and houses the statewide central database. The components deployed on the central Enterprise host are listed below:

- Transaction Support Services
- Compression
- Performance Logging
- Security
- Communications/COTS Services
- Data Validation/Consistency Checking
- Data Access and Referential Integrity
- Post Business Rule Services
- External Interfaces

Components Deployed on the Central Enterprise Host

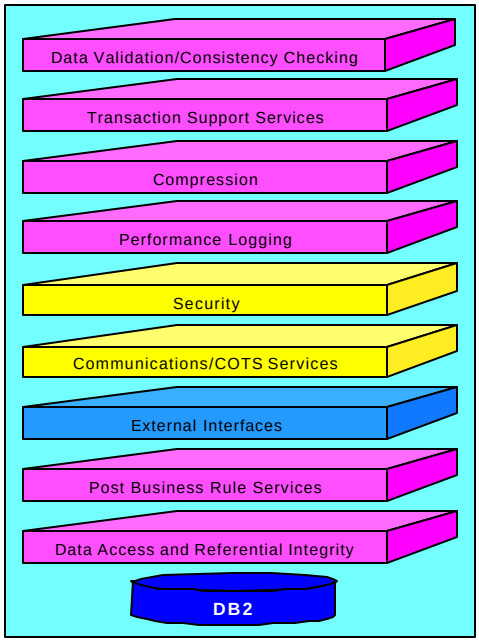


Figure 13: CWS/CMS Host Components

The Enterprise host acts as the application's SQL engine, providing the persistence for the application's information and the transaction support that extracts and packages the user's perspective of the statewide information for delivery to the desktop cache and presentation layer.

Several of the components or services shown above are ancillary to the Enterprise host's primary mission to provide access to persistent child welfare information. These components and service routines are important and will be discussed in the context and framework of the central platform's primary mission.

2.1.3.1. Transaction Support Services

This component relies on the services provided by CICS/Transaction Server for On-Line Transaction Processing (OLTP). CICS services and definitions manage connectivity and delivery mechanisms. Central CICS services are scaleable and redundant. They provide workload balancing and recovery services for the application.

The application architecture relies on these COTS services. The programs that run in the Enterprise host CICS regions use a variety of standard CICS services to perform their function.

CICS memory management is needed to contain response data during collection processing. Transaction response sizes can get large, and delivery can require multiple communication exchanges. CICS temporary storage management is also needed to contain response data after the CICS memory structures have been released. Error files contain codes and textual error descriptions. The majority of programs that execute on the Enterprise host platform are built from consistent template logic that is reliable and bug-free.

The transaction component consists of generated support for moving the information that is managed by the users between the presentation layer on the desktop and the persistent central database. Transactions originate at the desktop and represent the user's need to review and modify child welfare services information. Requests are issued and responses are returned.

A single network interface program manages the communication with the desktop CICS universal client using a standard CICS communication area structure referred to as a COMMAREA. This program manages the document transactions and provides interface to the transaction router program for the database transactions.

A single transaction router controls the sequence in which a transaction's underlying programs are invoked. The router uses definitions stored in Assembler tables that are artifacts from the generation process to control the sequence of Data Access Program (DAP) invocation. The router also invokes common service routines to validate database referential integrity and rules processing during CWS/CMS's **Save to Database** processing.

Several examples of transactions that are typically requested by a user are listed below:

- Log on and obtain the user's workload or caseload information
- Open a specific case to retrieve or modify its information
- Open a specific referral to retrieve or modify its information
- Search for a client that matches a set of user-defined criteria
- Search for a placement home matching a set of user-defined criteria
- Save the user's modifications to the database
- Obtain the information to create a local report

This component is robust and extendible.

2.1.3.2. Compression

This component provides encryption, decryption, compression, and decompression services to the Transaction Support Services components. The compression and decompression routines used by the application architecture transaction components reduce both LAN and WAN network load and improve network capacity management. This results in improved system management and reduces network costs.

2.1.3.3. Performance Logging

This component provides logging for statistical and error information occurring during online application transaction use. The information is useful for determining problems and is compiled to identify usage patterns and load statistics. The component consists of the service routine that is invoked by the Transaction Support Services to capture session statistical information and the collection and reporting procedures that consolidate and summarize daily activity.

2.1.3.4. Security

The service routines in this component interact with desktop requests to register and administer RACF user authorities to use the application. Once the user is defined to RACF, COTS products such as CICS and DB2 control access to system resources based on RACF authority levels assigned by the county administrator. (The application's data model defines the privileges and business rules used to restrict access to application functionality and to further control security.)

2.1.3.5. Communications/COTS Services

The Enterprise host uses COTS products and definitions to provide connectivity between and across the platforms in the technical architecture. CICS Transaction Server products provide facilities that balance the application load at the Enterprise host, and they encapsulate transaction activity inside the CICS universal client External Call Interface (ECI) using the Distributed Program Link (DPL) function and COMMAREA support. TX Series definitions provide remote destination information. DPL encapsulates the application-to-application transaction dialog in a COMMAREA and enables application protocol independence. This allows the LAN application traffic to be configured to support several protocols: NETBIOS, TCP/IP, or LU6.2.

The local server-to-Enterprise host communication over the WAN uses CICS Transaction Server Inter-System Communication (ISC) support for APPC LU6.2. This support limits the amount of connection and session definition and provides session concentration as it improves system performance and network security.

2.1.3.6. Data Validation/Consistency Checking

The application architecture's central platform provides minimal direct support for the components that provide these services; the majority of this functionality is located on the desktop platform. This choice avoids interactive patterns of use, minimizes the extent of late rule checking performed by the application, and enables the central server to streamline data access and persistence. However, the application's central data access and transaction layers do provide mechanisms to enhance the desktop's ability to provide the services.

Consistent packaging services standardize the description of the application's information. The syntax provided by Transaction Packet Descriptor (XPD) and Data Packet Descriptor's (DPD) metadata definitions enforce consistency across the platforms. The definitions capture the

database's optional and mandatory attribute information and represent database cardinality in a consistent form.

The central platform also provides service routines that enable custom enforcement of business rules requiring a statewide frame of reference. Business logic that enforces compliance to regulation and enforces the relational rules that manage the consistency of the programs' data are written on the central platform.

2.1.3.7. Data Access and Referential Integrity

The Enterprise host data access components contain the Structured Query Language (SQL) operations that retrieve and store information into the central DB2 database.

Collectively, the Enterprise host Data Access Programs (DAPs) perform database operations and package response rows on behalf of user requests. Individually--in open or retrieve mode--these programs fetch/select and package response rows from a specific database DB2 entity or view. Intrusive Save operations will cause individual DAPs to apply modifications to a specific entity row in the database. The type of operation is based on a user's row level change activity.

When DAPs are organized to retrieve information around the boundary of a specific instance of a referential object like "Case ABC", the DAPs are known as a "transaction". Transactions return referentially complete sets of data that represent the database content of a requested object such as a case. They assemble responses by sequentially controlling the invocation and packaging order of a series of DAPs. New transactions and DB2 entities are identified and defined during the project's design activities. Transactions define which entities are included in a transaction response, and DAPs are constructed to access the new entities.

Template-based code generation from COBOL language logic templates is used to build consistent data access logic. The generation process provides service routines to integrate custom SQL calls for data from the host database. The generation process also constructs other artifacts used in the Enterprise host online application to define transaction sequence and referential integrity checking. Service routines use the transaction sequence information to control DAP invocation order and to enforce Referential Integrity. A service routine constructs SQL checks that validate whether the Save to Database is referentially complete.

2.1.3.8. Post Business Rule Services

The Business Rules that cannot be enforced at the workstation because the necessary data is not available in the workstation cache are enforced at the host. The Post Business Rules Services component provides service routines to the Transaction Router that are executed after the data is modified in the database. These service routines perform business rule logic on the modified, deleted, and inserted records in the database to ensure data integrity. If the modified data does not confirm to the post business rules, then the entire database transaction is rolled back and the user receives an error.

2.1.3.9. External Interfaces

External interfaces are supported by calls to generated Data Access Programs (DAPS) that are triggered automatically by changes in the data. Therefore, when an update is made to CWS/CMS data in the database, it is automatically sent out to the appropriate location. Generally the automation consists of "queuing" the data to a staging area on the database and then processing the data in batch mode to create files for external distribution. The interfaces are proprietary (open, but not based on standards) and have been specifically designed and built for the application.

Appendix A — Basic Application Flow for Users

Overview

The CWS/CMS application is a full-function client/server tool that is used to do child welfare case management. Users log on to the network, which identifies them to the application. The application validates the user's authority to use the application and issues a mainframe request or transaction that returns a list of the user's caseload information. The startup sequence also delivers changes to control tables and any refreshes to the application itself.

1. A caseworker reviews the caseload information and then makes choices about which application to work in or which assignment to begin working on. When they select a specific unit of work (normally a Case or Referral), a request is sent to the mainframe to retrieve the database information surrounding their choice.

2. The response data is placed in the workstation's memory in a local database referred to as the "cache". The application architecture provides reusable service routines that enable the presentation layer to reference the database and populate controls and forms as the caseworker navigates within the application's GUI presentation.

3. A caseworker reviews the information that has been returned and performs any required action or needed activity. Their review of the selected Case might cause them to determine that information contained in another case or referral is of interest. If this data is not already in the cache database, another mainframe transaction retrieves it from the central database and unpacks the response data into the cache.

4. When a caseworker has completed working on each of their selected assignments, they request that the changes be saved. This "save to database" action causes new and modified rows to be packaged as a transaction and transmitted to the mainframe to file the modifications in the central database.

5. Caseworkers perform many actions and the application is full of function, but large transactions limit the amount of mainframe interaction needed in the performance of a caseworker's duties.

6. The problem domain contains a diversity of users accessing a comprehensive relational database in different fashions. The database contains child welfare information and this information varies significantly from case to case. The application's business data accumulates, and a case will vary from year to year.

7. Different users access information in different ways. Counties also use the application differently and have varying numbers of users and user roles.

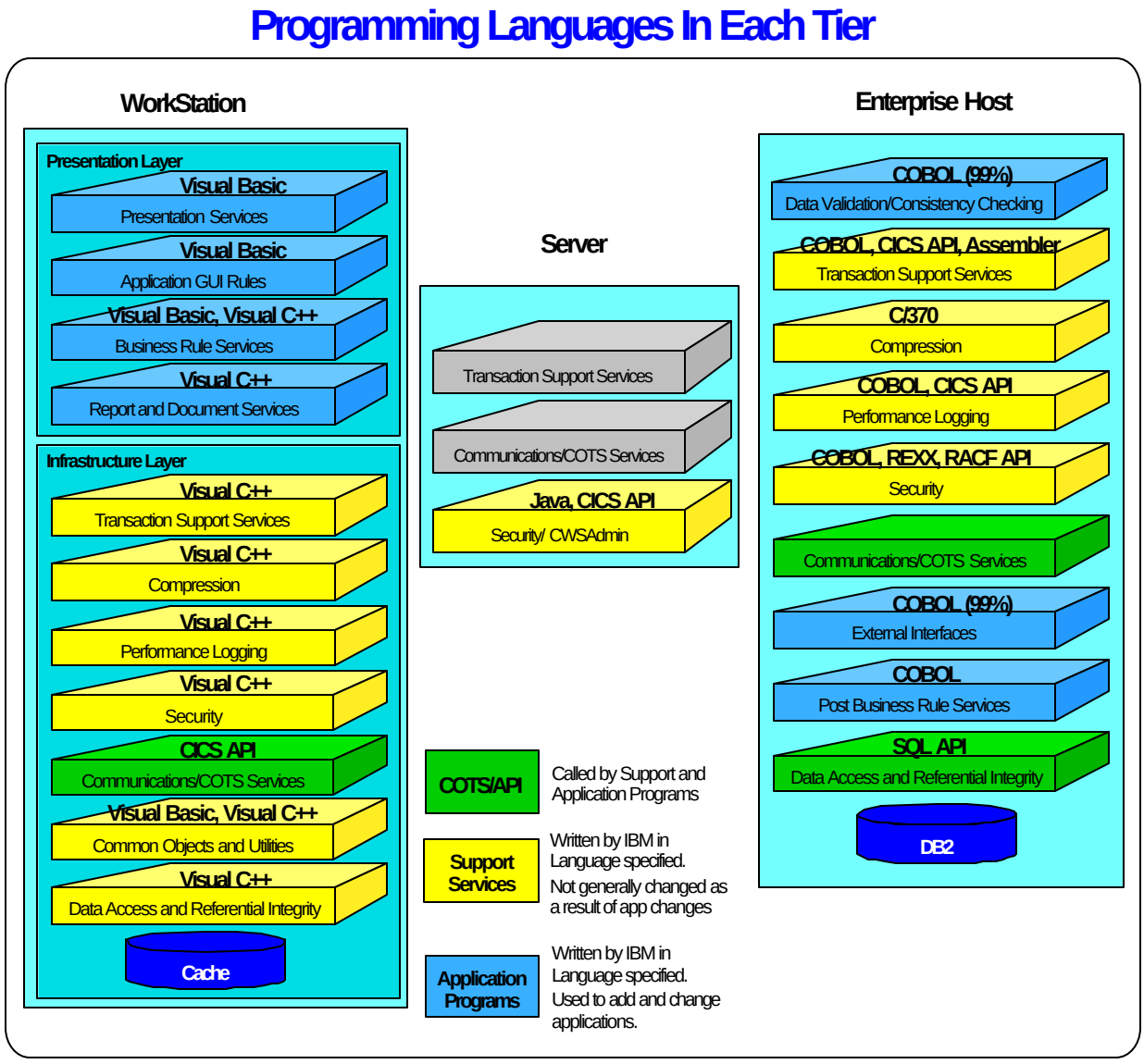
8. A data-driven application architecture addresses this problem domain, and a modeless user interface provides flexibility and full function to the application's users access to information.

Appendix B — CWS/CMS Programming Languages

This appendix is a short review of the programming languages used in the CWS/CMS system and it covers the influence they have on current and future system enhancements. In addition, this section reviews the effect that the development environment has upon the development and maintenance of the CWS/CMS application.

Overview

The system uses multiple languages to accomplish the goals for the CWS/CMS application. The languages used depend on the platform and are enhanced or “extended” by Application Programming Interfaces (APIs) used to interface to COTS components in the system. The following diagram illustrates where the languages are used in the system and where they interface using an API to the COTS modules.



Maintainability

The design point for the application programs for the CWS/CMS application included the construction of a significant amount of common infrastructure that was combined with COTS and general programming languages to form a "Programming Model" for CWS/CMS. The development of the programming model, particularly with the high reuse infrastructure modules, was intended to and is providing the following benefits:

- Reduces the amount of time required to introduce enhancements to the system by using code generators and commonly called reusable modules.
- Enforces programming standards by requiring application developers to build on existing layers and by disallowing their reinvention of data access and communications access routines. Note: the existing layers may be modified for performance or efficiency without rewriting additional layers as well.
- Reduces the impact of new application programs on the system by using previously tested and reliable common modules.
- Provides a common, single central data model and database that could be shared statewide and across multiple major applications.

Time Reduction for Enhancements

In essence, applications added to the CWS/CMS application that use the existing infrastructure can be built faster, with lower maintenance costs, and with greater reliability. With the introduction of Release 4.0 in late 1999, significant errors were introduced, a situation resulting mainly from the large amount of common programming module code that had to be reengineered and rewritten to move to the 32-bit Windows platform. Since this effort was completed, the major enhancements to CWS/CMS application have not resulted in significant injection of errors.

It is interesting to compare the level of change that was done in Release 4. In the past, many line-of-business systems were developed using IBM's flagship transaction monitor product CICS. Like today, these systems represented years of effort and were important to the constituents they served. At the time CICS was introduced, the interface used to support the CICS system was called "Macro-Level" CICS. That is, the APIs that supported various CICS functions used a particular API set, and the programs written in that environment were dependent upon those calls.

In later years, IBM developed a new and more powerful API set called "Command Level" CICS. For a number of releases, users were warned that the older level "Macro Level" calls would no longer be supported. Many users were unable or unwilling to take up the task of modifying their code base. Additionally, the newer calls required additional "horsepower" in the main processor to perform them, and they therefore created additional processing needs. As promised, IBM eventually suspended support for these calls in future releases. Many companies were forced to make a change to the new calls, and since the calls underpinned a majority of the functions performed by the systems, this change was large.

For the CWS/CMS application, the situation was analogous to the CICS support situation. Microsoft introduced the 32-bit APIs as a part of the original Windows 3.1.1 release, and later modified them when they introduced Windows 95 and Windows 98. After quite some time,

these APIs were enforced, especially when using products such as Microsoft Word, which is used by the CWS/CMS application for document processing. The project was forced to make the move to the 32-bit APIs, which were slower, needed more processor power, and removed some of the behaviors upon which the system was built. However, the path forward required the change and had to be taken. It is not surprising that a change of that magnitude came with a certain degree of difficulty. The CWS/CMS project has persevered during this transition, and release 5.0, release 5.2, and all the interim releases to-date have exhibited the type of stability with which users have become familiar.

In terms of the cost to build new applications, a substantial amount of the host data access is generated automatically based on specifications (as much as 90%), while the workstation applications reuse common routines that substantially reduce the programming effort. This allows developers to concentrate on business and not systems function, and it provides for functionality that can be introduced faster and with greater reliability. Equally important is the use of a central common data model and centralized database that frees the developer to work on business logic and not be distracted worrying about tedious details such as backup, recovery, and data synchronization.

Enforce Programming Standards

Because much of the additional programming for enhancements to the system is based in the workstation, the developers are forced to use existing “program templates” to complete their work. This includes templates that generate the data access code at the host, templates that illustrate how to use the existing “callable APIs” to get access to the data, and templates that illustrate how to use the existing APIs (e.g., to access a document to be created in the built-in COTS word processor). This also helps enforce standard naming conventions, which makes follow-up maintenance programming easier.

By using a common data model, the developer is forced to use data names that are common across the system for accessing data; in addition, they are forced to use generated routines to access the data.

Because much of the tedious programming work (data access etc) is done for the developer, there is a tendency to use the standards reflexively, which makes it easier to build new modules (proven routines are used), and the code is reused (common modules).

Reduce the Impact

Reuse of existing modules that have been tested previously is a well-founded software engineering technique. The key in reuse is to reduce the impact of new code because:

- The programming statements are previously written, which reduces the amount of labor needed to write new program statements.
- Reused modules are proven both by previous testing and field use, which helps to ensure a more reliable new component and makes testing and rework conclude quickly.

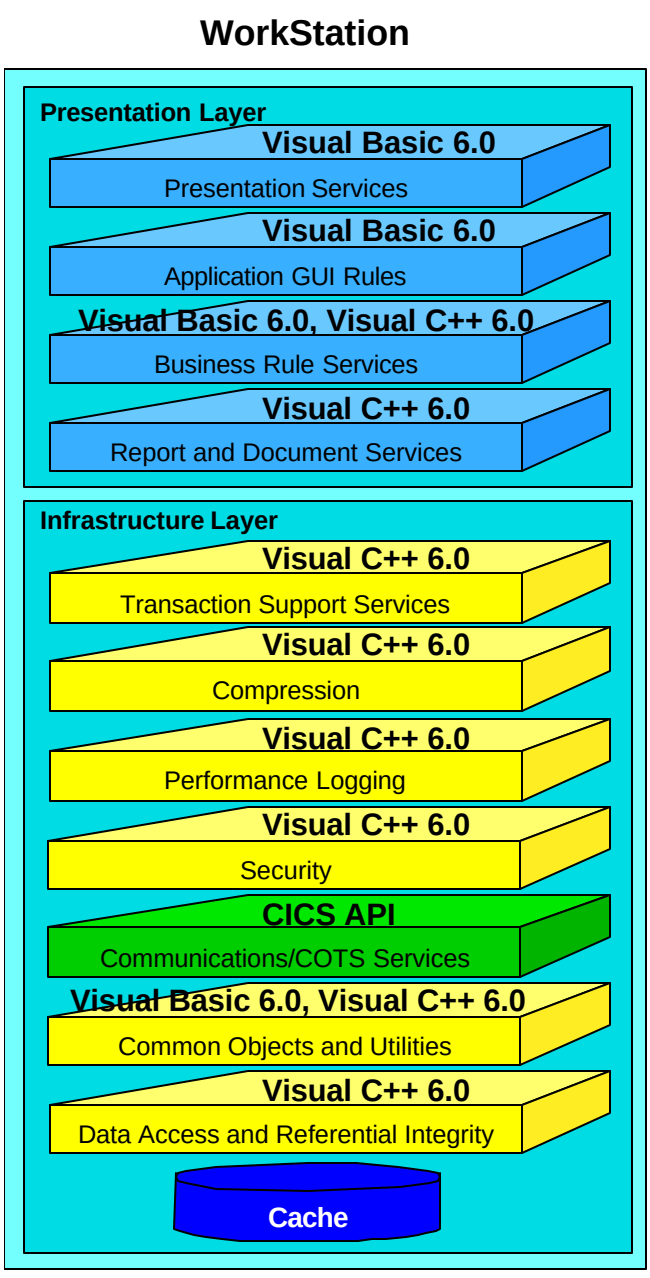
In the CWS/CMS application, this reuse is supplemented with the automated generation of data access modules at the host, which also makes testing conclude faster and with less rework.

Common Data Model and Central Database

The common data model allows developers to progress faster. This is because data element names are used consistently across all programs (also reducing debugging), thereby increasing the developers' absorption of new specifications. This works for the requirement gathering and program development.

Due to its standardization, the centralized database provides for a common programming model that is easily understood by the programming teams. The larger positive impact is the reduction in programming provided because difficult aspects of data update (like locking, recovery, roll back, etc.) are already articulated, designed, and largely automatically generated for the developer.

The single biggest attribute of the centralized database is that there is no requirement to synchronize data with other files because there is only one database in the system. This substantially reduces:



- The need for additional programming (no synchronization routines are required)
- The need for additional testing (there are no synchronization routines to test)
- The need for overly complex data recovery logic (only one database is involved, not multiples to coordinate)

And finally, a bonus is provided to CWS/CMS's future. Because of the separation of the data store routines on the host from the front-end processing on the workstation, other systems can access and update the central database without having to use the identical front-end programming tools. This allows CWS/CMS to expand its horizons with new user tools that can be implemented without complex reprogramming and data coordination.

Programming Languages

CWS/CMS currently is broken down into three distinct "platforms" for the delivery of function. The following is a discussion of the issues related to each platform with respect to the programming languages they use.

Workstation

The workstation uses Visual Basic 6.0 for screen displays and business editing logic and uses C++ 6.0 for the common modules that are reused. Both languages are mature, well understood, and have a widely available resource pool of trained personnel to draw from in the industry. In addition, direct access to a built-in COTS word processor is provided for the creation of documents managed by the CWS/CMS application.

The development environment for the workstation consists of reusable libraries stored on a central server, workstations with Visual basic 6.0 and C++ 6.0, and a Version control system at the central server that manages multiple versions of the application. Workstations are connected to a test Host system so that unit testing can be effectively performed. A separate group then performs testing with test workstations, using both the Windows 95 and Windows 2000 operating systems.

Visual Basic 6.0

Visual Basic 6.0 is used to build the screens displayed to the user (presentation services) and for edit logic on the fields in those screens (Application GUI Rules).

Visual Basic 6.0 Strengths

For extension of the current application, where reuse of the standard platform is the business necessity, VB 6.0 is the current leading choice. While other languages could be introduced, some infrastructure would have to be retooled to handle the need and there is no current justification for that

Visual Basic 6.0 Issues

The language is largely proprietary and limited to fat client workstations; this makes using it on thin clients prohibitive.

The language is not portable to scaleable servers, which makes it useful only as a workstation tool and not a cross-platform tool.

Communications to COTS routines is frequently done only using interface routines hand-built in C++ 6.0, which makes adding new COTS routines more difficult.

C++ 6.0

C++ 6.0 is used for the infrastructure routines and for interfacing to COTS and the operating system. Because of the tight coupling necessary for operating system (OS) interfaces, changing the OS has major implications such as those seen in the transition to Windows 32-bit. In that case, the Microsoft product was not upwardly compatible and a great deal of change was necessary.

C++ 6.0 Strengths

C++ 6.0 is in wide commercial use and is well understood. It provides access to low level operating system components and is efficient for execution, which reduces response time to users. It can be used to provide programming for multiple platforms. For example, in CWS/CMS, it is used both the Workstation and the Communications Server.

C++ 6.0 Issues

By necessity C++ 6.0 is tied to the operating system, which makes portability and conversion more complex when going to new operating systems or other platforms.

COTS Applications

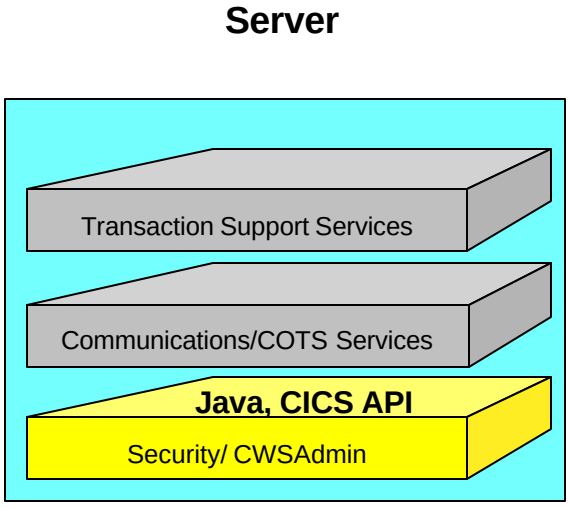
The COTS CICS API is used to communicate to the host database. It simplifies programming by providing all of the necessary internal communications and recovery routines necessary to effectively transfer and receive data from the central database. The desktop COTS word-processor is used by the application to produce reports and accept large amounts of text input. Templates are provided, easing the burden for the user.

COTS Strengths

The COTS routines extend the programming languages without requiring developers to develop all of the code required. It effectively reduces the number of managed function points in the system because it is integrated as a pre-tested “macro”.

COTS Issues

The COTS routines must be upgraded often because of support issues from the vendor. The cost of the upgrade includes every desktop, in contrast to program code that is written once for a single cost and distributed to all users at no additional charge.



Server

The server component of the CWS/CMS application routes workstation traffic and requests to and from the central database. It is used primarily to concentrate and interleave requests to the mainframe so that the individual application programs do not have to employ the complex code required to perform this task.

The development environment for the Server consists of reusable libraries stored on a central server, workstations with Java, and a version control system (PVCS) at the central server that manages multiple versions of the application. Workstations are connected to a test host system so that unit testing can be effectively performed. The CWS/CMS test team then tests it using Windows 95 and 2000.

Java

Java Strengths

In wide commercial use and well understood, Java is a purer Object Oriented Language and easier to use than C++. Java applications can be written once and run on many different platforms without any modifications. The CWSAdmin is developed in Java.

Java Issues

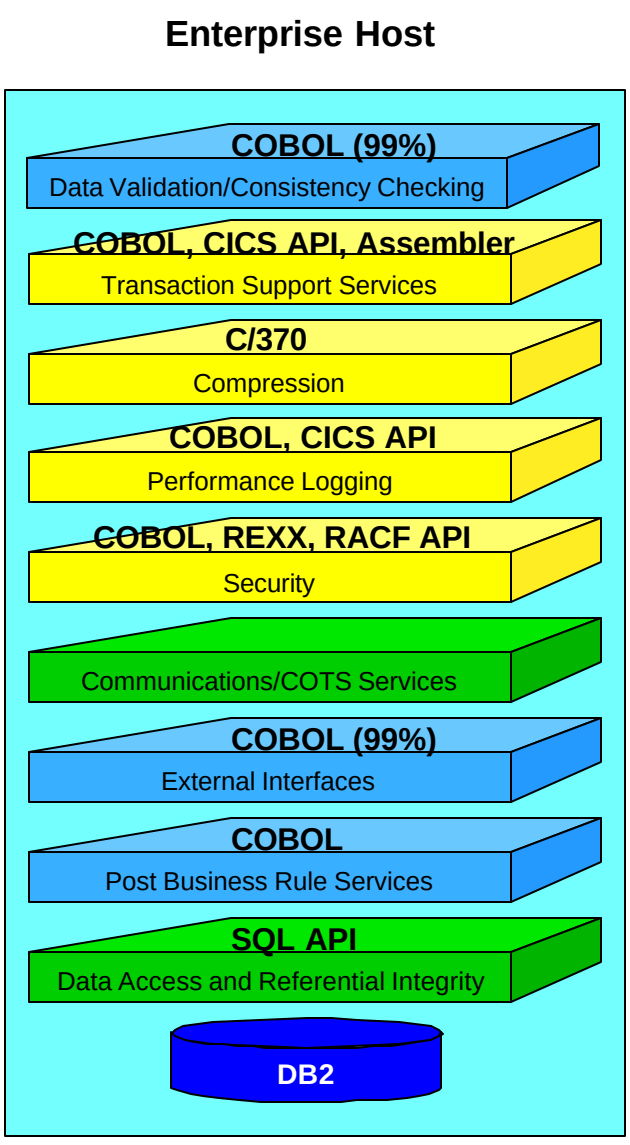
Java has rapidly had three different versions (1.0, 1.1, 1.2, 1.3, and 1.4) with important changes in each version. Deploying Java applications is more complex than C++ and Visual Basic applications.

COTS Applications on the Server

The COTS applications on the server dramatically reduce the amount of code that must be written for the application. These routines handle the tedious technical work of managing communications and transactions. What is important about these routines is that other workstation applications can use them as well as the current CWS/CMS workstations, making them flexible for expansion.

COTS Strengths

The COTS routines extend the programming languages without requiring developers to create all of the required programming statements. It reduces the number of managed function points in the system because it is integrated as a pre-tested "macro". The choice of technology in this case (CICS) has a consistent programming API across multiple platforms, making it both portable and flexible in an n-tier environment such as the CWS/CMS system.



COTS Issues

The use of COTS on the server has minimal weaknesses because of its flexibility.

Enterprise Host

The Enterprise Host uses primarily COBOL and COTS software to store and ensure the integrity of the CMS/CWS application's data. The data is stored in the Enterprise Host DB2 Table using the SQL API for storage and retrieval of information.

The development environment for the Enterprise Host consists of the developer's workstation, the host-based compilers, and a version control system to manage the different release levels. When changes are made to the data validation and storage routines, the changes are generated using the assembler routines.

COBOL

COBOL is used for the data validation routines and then makes SQL calls to store data.

COBOL Strengths

COBOL is a mature, well-understood language with a wide body of available developers who understand its use in mission-critical, high-availability applications. An English-like program, COBOL statements are easy to track back to business rules for data validation. Rules can be ported to multiple platforms. However, at the inception of CWS/CMS this was not available, so this approach has not been used.

COBOL Issues

COBOL was not available as a workstation language during the original workstation level development of the application. Consequently, it was not recommended for use in the workstation.

Other Languages

The other languages in use in the CWS/CMS system are Assembler (used for code generation and one minor routing program), C/370, and REXX (used in the Security component).

Other Language Strengths

These other languages are specialized for the functions in the CMS/CWS system, allowing the generation of code (Assembler), which simplifies changes to the database schema. The C/370 and REXX code is used for the security routines, where the C/370 code was the same as that used in the workstation and server components.

Other Language Issues

The minimal use of these languages for their specialized functions creates minimal impact.

Appendix C — Structure for Performance Logging

	POSITION	
RECORD ID	1	
SERVER	2 - 9	
USER ID	10 - 17	
COMPUTER NAME	18 - 25	
CORRELATION ID	26 – 35	
TRANSACTION ID	36 - 39	
START TIMESTAMP	40 – 59	YYYYMMDDHHMMSSNNNNNN
END TIMESTAMP	60 – 79	YYYYMMDDHHMMSSNNNNNN
ELAPSED TIME	80 – 89	
MESSAGE SIZE	90 - 99	
CACHE SIZE	100- 109	

Program on the host formats the data and stores it in a DB2 table. The structure of the DB2 table is given below.

```
CREATE TABLE PERFLOGT
   (PART_DAY CHAR(2)                          NOT NULL          ,
    START_TS TIMESTAMP                        NOT NULL          ,
    USER_ID  CHAR(8)                          NOT NULL          ,
    END_TS   TIMESTAMP                        NOT NULL          ,
    SERVER   CHAR(8)                          NOT NULL          ,
    COMP_NM  CHAR(8)                          NOT NULL          ,
    CORR_ID  CHAR(10)                         NOT NULL          ,
    TRAN_ID  CHAR(4)                          NOT NULL          ,
    ELAPSED  CHAR(10)                         NOT NULL          ,
    MSG_SIZE CHAR(10)                         NOT NULL          ,
    CACHE_SZ CHAR(10)                         NOT NULL          )
```

Appendix D — Release 5.0 Notes

Release 5.0 was designed to support both a Windows 95 client with an OS/2 or Windows 2000 server as well as a Windows 2000 client with a Windows 2000 server. The code changes made to support a Windows 2000 environment were done in a way that will minimize adverse performance impact and maximize code maintainability.

No new *application* functionality was added as a result of this release. The changes made to support both a Window 95 and Windows 2000 client entailed the rewriting of components to support a newer version of the Windows operating system (Windows 2000) with backwards compatibility with Windows 95.

The long-term viability of Release 5.0 with Windows 95 depends solely on the availability of support for Windows 95. Microsoft has announced they will discontinue support of the Windows 95 operating system on December 31, 2001. While there is no technical reason that Release 5.0 will not continue to operate with Windows 95 past this date, the availability of workstation hardware (either new or replacement components) will be in significant jeopardy. As a result, application support on the Windows 95 platform for the Release 5.0 (as well as any future releases) will not be possible beyond Microsoft's published end of support for Windows 95.

The Windows 2000 Server Infrastructure project (WA0005A) details the infrastructure and server architecture changes resulting from the overall server operating system migration to Windows 2000. Refer to the (WA0005A) *CWS/CMS Infrastructure Architecture* and *CWS/CMS Server Architecture* documents for details on this migration.

Release 5.0 Changes

To better understand the changes resulting from the application port, a brief review of how the workstation talks to the server is necessary. There are two pieces of code that communicate with the application server. The first, the CICS Workstation Client, is responsible for managing transactions that are run to the host. The second, PipeMan, is responsible for communicating with CWSAdmin. To support Windows 2000 and correctly handle security with the Windows 2000 server, the CICS Workstation Client was upgraded to Version 3.1.2.

PipeMan had the more interesting changes. When PipeMan is loaded, it determines the type of server to which the workstation is connected by examining the format of the name of the server. If the workstation is connected to an OS/2 server, the code operates as it always did and communicates with the server using named pipes. If the workstation is connected to a Windows 2000 server, a new code path is taken and sockets are used.

Performance logging was also affected by the introduction of Windows 2000 servers. To improve the efficiency of performance data collection, data is logged directly to the host. This new logging mechanism uses the transaction infrastructure of CWS/CMS. The technical details of Performance logging are described in more detail in the *CWS/CMS Server Architecture Review* section of this document. Note that while the logging mechanism has been changed, neither the type of information being collected nor the telemetry points (monitoring locations) were modified in this release.

Performance measurements performed on Release 5.0 are documented in the *Release 5.0 Test Exit Report* (WA0007 – Deliverable #B4) and the *Windows 2000 Architecture Test Exit Report* (WA005A – Deliverable #5). As with any new release, the previous release's

performance measurements (as documented in *Release 4.2 Final Test Exit Report* WA0002-2) were used as a baseline for comparison.

The following table lists the workstation Dynamic Link Library (DLL) changes as a result of the Release 5.0 application port. No other COTS components were modified or updated.

Added in 5.0	Newer in 5.0	Removed in 5.0
ADVPACK.DLL	ASYCFILT.DLL	MFC42D
ATL.DLL	COMCAT.DLL	MFCO42D
MFC42U.dll	MFC42.DLL	MSVCIRTD
MSVBVM60.DLL	MSVCIRT.DLL	MSVCRTD
MSVCP60.DLL	MSVCRT	
	OLEAUT32	
	OLEPRO32.DLL	

There were no changes to the OS/2 server workstation components for Release 5.0. The following components are used for Windows 2000 servers:

Windows 2000 Server Components for Release 5.0
TX-Series
CICS Universal Client
IBM Communication Server
CWSAdmin
Java Runtime Environment
CICS Transaction Gateway